数据接入服务

SDK 参考

文档版本 01

发布日期 2024-10-28

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 简介

1.1 DIS SDK 能做什么

DIS 概述

数据接入服务（Data Ingestion Service）为处理或分析流数据的自定义应用程序构建数据流管道，主要解决云服务外的数据实时传输到云服务内的问题。数据接入服务每小时可从数十万种数据源（如日志和定位追踪事件、网站点击流、社交媒体源等）中连续捕获、传送和存储数TB数据。

云服务实现了在多地域部署基础设施，具备高度的可扩展性和可靠性，用户可根据自身需要指定地域使用DIS服务，由此获得更快的访问速度和实惠的服务价格。

DIS对数据传输所需要的基础设置、存储、网络和配置进行管理。您无需为数据通道担心配置、部署、持续的硬件维护等。此外，DIS还可在云区域同步复制数据，为您提供数据高可用性和数据持久性。

SDK 概述

数据接入服务软件开发工具包（DIS SDK，Data Ingestion Service Software Development Kit）是对DIS服务提供的REST API进行的封装，以简化用户的开发工作。用户直接调用DIS SDK提供的接口函数即可实现使用DIS服务业务能力的目的。

1.2 内容导航

SDK开发指南指导您如何安装和配置开发环境、如何通过调用DIS SDK提供的接口函数进行二次开发。

章节	内容
DIS SDK能做什么 内容导航	简要介绍DIS的概念和DIS SDK的概念。

章节	内容
SDK下载 **兼容性** **如何校验软件包完整性?**	介绍使用DIS SDK进行二次开发过程中涉及到的资源信息。
开通DIS服务	介绍DIS服务和DIS通道的开通方式。
获取认证信息	介绍使用DIS SDK进行二次开发前需要进行的初始化工作。
Java：**准备环境**~~**变更分区数量**	介绍使用DIS SDK进行的常用操作（匹配java）。
DIS服务端错误码	介绍使用DIS SDK过程中遇到异常时的响应信息。

2 相关资源

2.1 SDK 下载

在**https://github.com/huaweicloud/huaweicloud-sdk-java-dis**中下载DIS的Java SDK压缩包。

获取DIS SDK软件包及校验文件后，需要对软件包的完整性进行校验，参考**如何校验软件包完整性？**。

2.2 兼容性

支持的JDK版本：1.8.0及以上版本。

支持的Python版本：2.7及以上版本。

2.3 如何校验软件包完整性?

获取DIS SDK软件包及校验文件后，可以在Linux系统上按如下步骤对软件包的完整性进行校验。

前提条件

- 已获取“PuTTY”工具。
- 已获取“WinSCP”工具。

操作步骤

步骤1 使用“WinSCP”工具将“huaweicloud-sdk-dis-x.x.x.zip”上传至Linux系统任一目录。

说明

x.x.x表示DIS SDK包的版本号。

步骤2 使用“PuTTY”工具登录Linux系统，进入到“huaweicloud-sdk-dis-x.x.x.zip”所在目录，执行如下命令，获取DIS SDK压缩包的校验码。

sha256sum huaweicloud-sdk-dis-x.x.x.zip

显示类似如下校验码：

```
# sha256sum dis-sdk-x.x.x.zip
8be2c937e8d78b1a9b99777cee4e7131f8bf231de3f839cf214e7c5b5ba3c088 huaweicloud-sdk-dis-x.x.x.zip
```

步骤3 打开DIS SDK的校验文件“huaweicloud-sdk-dis-x.x.x.zip.sha256sum”与上一步骤中获取的校验码进行对比。

- 一致，说明从获取的DIS SDK压缩包没被篡改。
- 不一致，说明DIS SDK压缩包被篡改，需要重新获取。

----结束

3 开通 DIS

3.1 开通 DIS 服务

步骤1 注册云服务账号。

步骤2 开通DIS服务。

使用DIS服务之前必须先充值，才能正常使用DIS服务。

1. 登录DIS管理控制台。
2. 单击页面右上角的“费用”。
3. 单击“充值”，系统自动跳转到充值窗口。
4. 根据界面提示信息，对账户进行充值。
5. 充值成功后，关闭充值窗口，返回管理控制台首页。
6. 单击“数据接入服务”，开通服务。

步骤3 创建访问密钥。

DIS通过用户账户中的AK和SK进行签名验证，确保通过授权的账户才能访问指定的DIS资源。

1. 登录DIS控制台。
2. 单击页面右上角的用户名，选择“我的凭证”。
3. “我的凭证”页面，单击“管理访问密钥”区域下方的“新增访问密钥”。
4. 根据界面提示输入相关信息并保存新创建的访问密钥。

说明

- 每个用户最多可创建两个有效的访问密钥。
- 为防止访问密钥泄露，建议您将其保存到安全的位置。如果用户在此提示框中单击“取消”，则不会下载密钥，后续也将无法继续下载，用户必须将此密钥删除后再创建新的访问密钥。

----结束

4 开通 DIS 通道

开通数据接入服务的请参见《 数据接入服务用户指南 》中的“开通DIS通道”。

具体操作请参见**开通DIS通道**。

5 获取认证信息

获取访问密钥

您可以通过如下方式获取访问密钥。

1. 登录控制台，在用户名下拉列表中选择“我的凭证”。
2. 进入“我的凭证”页面，选择“访问密钥 > 新增访问密钥”，如图5-1所示。

图 5-1 单击新增访问密钥

3. 单击“确定”，根据浏览器提示，保存密钥文件。密钥文件会直接保存到浏览器默认的下载文件夹中。打开名称为“credentials.csv”的文件，即可查看访问密钥（Access Key Id和Secret Access Key）。

说明

- 每个用户仅允许新增两个访问密钥。
- 为保证访问密钥的安全，访问密钥仅在初次生成时自动下载，后续不可再次通过管理控制台界面获取。请在生成后妥善保管。

获取项目 ID 和账号 ID

项目ID表示租户的资源，账号ID对应当前账号，IAM用户ID对应当前用户。用户可在对应页面下查看不同Region对应的项目ID、账号ID和用户ID。

1. 注册并登录管理控制台。
2. 在用户名的下拉列表中单击“我的凭证”。
3. 在“API凭证”页面，查看账号名和账号ID、IAM用户名和IAM用户ID，在项目列表中查看项目和项目ID。

获取区域和 Endpoint 信息

终端节点（ Endpoint ）即调用API的请求地址，不同服务不同区域的终端节点不同。DIS的终端节点Endpoint构造规则为dis.{region_id}.{域名}，您可以从**地区和终端节点**获取区域和终端节点信息。

6 使用 SDK

6.1 使用 SDK（Java）

6.1.1 准备环境

- 已从**Oracle官网**下载并安装JDK1.8或以上版本，配置好JAVA环境变量。
- 已从**Eclipse官网**下载并安装Eclipse IDE for Java Developers最新版本。
- 已在Eclipse中配置好JDK。

6.1.2 配置样例工程

SDK下载的“huaweicloud-sdk-dis-java-*X.X.X*.zip ”最新版本压缩包中提供了SDK源码；**DIS SDK桶**中的“huaweicloud-sdk-dis-java-*X.X.X*.zip ”最新版本压缩包提供了示例工程，您可以在本地设备上使用开发工具（如Eclipse）编译运行示例工程，或以示例工程为基础开发您的应用。示例工程代码路径：“\dis-sdk-demo\src\main\java\com\bigdata\dis\sdk\demo”。

示例代码	说明
ConsumerDemo.java	展示了下载数据的用法
ProducerDemo.java	展示了上传数据的用法

操作步骤

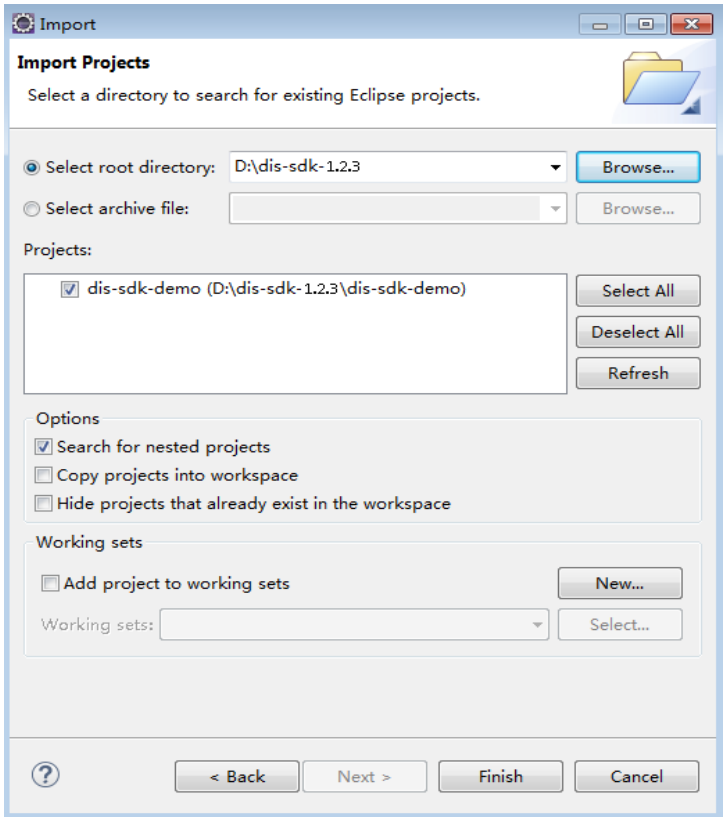
步骤1 解压**DIS SDK桶**中的“huaweicloud-sdk-dis-java-*X.X.X*.zip ”压缩包获得dis-sdk-demo包。

步骤2 导入Eclipse项目。

1. 打开Eclipse。选择“File > Import”弹出“Import”窗口。
2. 选择“Maven > Existing Maven Projects”，单击“Next”，进入“Import Maven Projects”页面。

3. 单击“Browse”按钮，根据实际情况选择“dis-sdk-demo”样例工程的存储位置，勾选样例工程，如图6-1所示。

图 6-1 Import Maven Projects

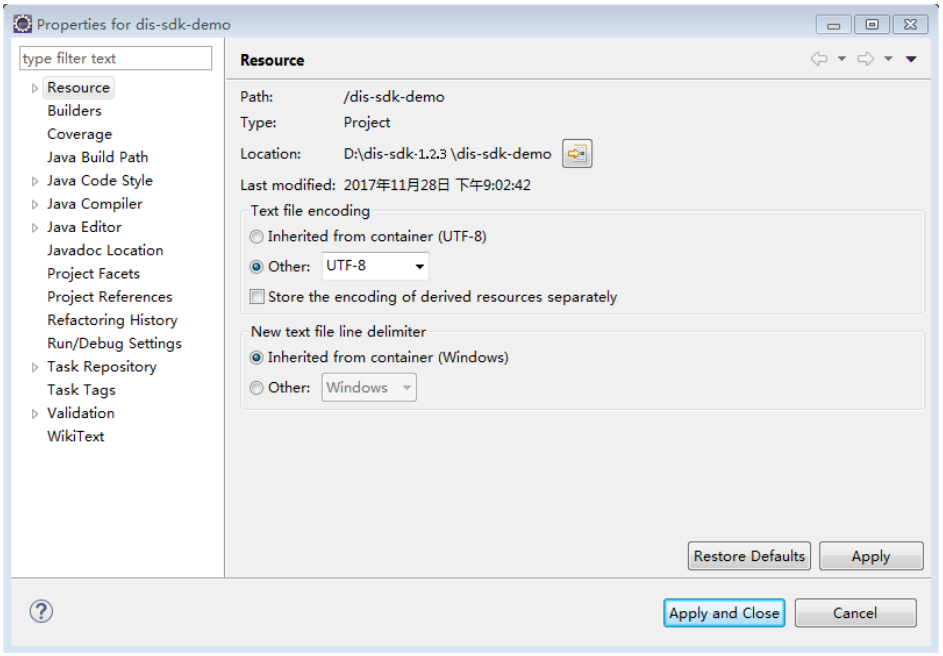


4. 单击“Finish”完成项目导入。

步骤3 配置Demo工程。

1. 配置项目编码为“UTF-8”。
   a. 在左侧导航栏“Project Explorer”中右键单击所需工程，选择“Properties ”，进入“Properties for dis-sdk-demo”页面。
   b. 左侧页签栏选择“Resource”，右侧对话框显示“Resource”页面。
   c. 在“Text file encoding”栏中选择“Other”，单击下拉框选择“UTF-8”。
   d. 单击“Apply and Close”完成编码配置。

图 6-2 Resource



2. 添加JDK。
   a. 在左侧导航栏“Project Explorer”中右键单击所需工程，选择“Properties”，进入“Properties for dis-sdk-demo”页面。
   b. 左侧页签栏选择“Java Build Path”，右侧对话框显示“Java Build Path”页面。
   c. 在“Java Build Path”页面选择“Libraries”页签，单击“Add Library”，弹出“Add Library”对话框。
   d. 选择“JRE System Library”，单击“Next”确认“Workspace default JRE”为jdk1.8及以上版本。
   e. 单击“Finish”退出“Add Library”对话框。
   f. 单击“Apply and Close”完成JDK添加。

----结束

6.1.3 初始化 DIS 客户端

您可以使用以下两种方法初始化DIS SDK客户端实例，优先选择使用代码进行初始化。其中，“endpoint”，“ak”，“sk”，“region”，“projectId”信息请参考**获取认证信息**。

- 使用代码初始化DIS SDK客户端实例。

```
// 创建DIS客户端实例
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
     // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
    // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
    .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    // 以下配置失败时的重试次数
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
```

```
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```

- 若需要使用代理，请使用如下方法初始化DIS客户端：

```
// 创建DIS客户端实例
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
     // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
    // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
    .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    .withProxyHost("YOUR_PROXY_HOST") // 代理IP
    .withProxyPort("YOUR_PROXY_PORT") // 代理端口
    .withProxyProtocol(Protocol.HTTP) // 代理协议，默认为HTTP
    .withProxyUsername("YOUR_PROXY_USER_NAME") // 代理用户名（可选）
    .withProxyPassword("YOUR_PROXY_PASSWORD") // 代理密码（可选）
    // 以下配置失败时的重试次数
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```

- 若需在客户端设置DIS连接超时时间，请使用如下方法初始化DIS客户端：

```
// 创建DIS客户端实例
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
    .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
    .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    .withProperty(DISConfig.PROPERTY_CONNECTION_TIMEOUT, "60")  // 单位：秒
    .build();
```

- 若需要开启传输压缩，请使用如下方法初始化DIS客户端：

```
// 创建DIS客户端实例
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
    .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
    .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    .withBodyCompressEnabled(true)
    .withBodyCompressType(CompressionType.ZSTD) // 配置压缩算法，当前支持lz4和zstd，默认值为lz4
    // 以下配置失败时的重试次数
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```

- 若需在客户端将数据加密后再上传到DIS，DIS SDK提供了加密方法。即在构建disclient时增加参数DataEncryptEnabled和data.password。

```
// 创建DIS客户端实例
DIS dic = DISClientBuilder.standard()
    .withEndpoint("YOUR_ENDPOINT")
    .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
    .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
    .withProjectId("YOUR_PROJECT_ID")
    .withRegion("YOUR_REGION")
    .withDataEncryptEnabled(true)
    .withProperty("data.password", "xxx") //xxx替换为用户配置的数据加密密钥
    // 以下配置失败时的重试次数
    .withProperty(DISConfig.PROPERTY_PRODUCER_RECORDS_RETRIES, "-1")
    .withProperty(DISConfig.PROPERTY_PRODUCER_EXCEPTION_RETRIES, "-1")
    .build();
```

说明

若使用JAVA SDK加密上传数据，读取数据也需要使用JAVA SDK配置相同的密钥。

- 使用配置文件初始化DIS SDK客户端实例。

  在“dis-sdk-demo\src\main\resources”目录下的“dis.properties”文件中添加如下配置项。

  - ak/sk：用户在IAM中创建的AK/SK。
  - region：用户使用通道所在的区域。
  - endpoint：DIS的访问地址。
  - projectId：通道所在的资源ID。

```
// 创建DIS客户端实例
        DIS dic = DISClientBuilder.standard().build();
```

6.1.4 创建通道

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK创建DIS通道，需要指定通道名称、通道的分片数量以及通道类型。

其中，普通通道为“STREAM_TYPE_COMMON”，高级通道为“STREAM_TYPE_ADVANCED”。

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
// 通道名称
String streamName = "myStream";
createStreamRequest.setStreamName(streamName);
// 通道类型: COMMON 普通通道; ADVANCED 高级通道
createStreamRequest.setStreamType(StreamType.COMMON.name());
// 通道的分片数量
createStreamRequest.setPartitionCount(3);
// 通道数据的保留时长: 单位小时，N*24，N的取值为1~7的整数
createStreamRequest.setDataDuration(24);
// 通道的源数据类型：缺省值：BLOB
createStreamRequest.setDataType(DataTypeEnum.BLOB.name());
```

配置“CreateStreamRequest”对象之后，通过调用createStream的方法创建通道。

```
dic.createStream(createStreamRequest);
```

6.1.5 添加转储任务

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK创建转储任务，需要指定通道名称、转储任务名称，转储周期，转储目标服务等信息。

添加转储到对象存储服务（OBS）的转储任务

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

// 配置通道名称：用户在数据接入服务(简称DIS)控制台创建通道
request.setStreamName(streamName);

// 添加OBS转储任务，并设置任务名称
OBSDestinationDescriptorRequest descriptor = new OBSDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

// 转储至对象存储服务(简称OBS)：OBS桶名和子文件夹名，通过OBS控制台或客户端创建桶和文件夹
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

// 转储周期，单位s
descriptor.setDeliverTimeInterval(900);
```

```
// 可选：在DIS管理页面自动创建名称为“dis_admin_agency”的IAM委托，默认采用此委托，用于授权访问。如未创建过IAM委托，请用主账户登录DIS控制台并创建通道，点击“添加转储任务”，前往授权。
descriptor.setAgencyName("dis_admin_agency");

// 可选，转储OBS的目标文件格式：默认text，可配置parquet、carbon
descriptor.setDestinationFileType(DestinationFileTypeEnum.TEXT.getType());

// 设置从DIS通道拉取数据时的初始偏移量: 默认LATEST，从通道内最新上传的记录开始读取; TRIM_HORIZON，从通道内最早的未过期记录开始读取
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setObsDestinationDescriptor(descriptor);
```

配置“CreateTransferTaskRequest”对象之后，通过调用createTransferTask的方法创建转储任务。

```
dic.createTransferTask(request);
```

添加转储到 MapReduce 服务（ MRS ）的转储任务

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//配置通道名称：用户在数据接入服务(简称DIS)控制台创建通道
request.setStreamName(streamName);

//添加MRS转储任务，并设置任务名称
MRSDestinationDescriptorRequest descriptor = new MRSDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

// 配置MRS集群信息：集群名称和集群ID。可通过弹性大数据服务(简称MRS)控制台创建和查询，集群需为非安全模式
descriptor.setMrsClusterName("mrs_dis");
descriptor.setMrsClusterId("fe69a732-c7d3-4b0f-8cda-ec9eca0cf141");

// 转储MRS通过OBS服务中转，需配置OBS桶名和子文件夹名，此目录也用于保存转储失败的源数据文件。可通过OBS控制台或客户端创建桶和文件夹
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

// 转储周期，单位s
descriptor.setDeliverTimeInterval(900);

// 可选：在DIS管理页面自动创建dis_admin_agency委托后，默认采用此委托。如未创建过IAM委托，请用主账户登录DIS控制台并创建通道，点击“添加转储任务”，前往授权。
descriptor.setAgencyName("dis_admin_agency");

// 转储OBS的目标文件格式：默认text，可配置parquet、carbon
descriptor.setDestinationFileType(DestinationFileTypeEnum.TEXT.getType());

// 设置从DIS通道拉取数据时的初始偏移量: 默认LATEST，从通道内最新上传的记录开始读取; TRIM_HORIZON，从通道内最早的未过期记录开始读取
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setMrsDestinationDescriptor(descriptor);
```

配置“CreateTransferTaskRequest”对象之后，通过调用createTransferTask的方法创建转储任务。

```
dic.createTransferTask(request);
```

添加转储到数据湖探索服务（ DLI ）的转储任务

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

// 配置通道名称：用户在数据接入服务(简称DIS)控制台创建通道
request.setStreamName(streamName);

// 添加DLI转储任务，并设置任务名称
```



```
UqueryDestinationDescriptorRequest descriptor = new UqueryDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

// 配置DLI相关信息：数据库和内表名称。可通过数据湖探索(简称DLI)控制台创建和查询，DLI表需为内表
descriptor.setDliDatabaseName("dis_dli");
descriptor.setDliTableName("dis_test");

// 转储DLI通过OBS服务中转，需配置OBS桶名和子文件夹名，此目录也用于保存转储失败的源数据文件。可通过OBS控制台或客户端创建桶和文件夹
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

// 转储周期，单位s
descriptor.setDeliverTimeInterval(900);

// 可选：在DIS管理页面自动创建dis_admin_agency委托后，默认采用此委托。如未创建过IAM委托，请用主账户登录DIS控制台并创建通道，点击“添加转储任务”，前往授权。
descriptor.setAgencyName("dis_admin_agency");

// 设置从DIS通道拉取数据时的初始偏移量: 默认LATEST，从通道内最新上传的记录开始读取; TRIM_HORIZON，从通道内最早的未过期记录开始读取
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setDliDestinationDescriptor(descriptor);
```

配置“CreateTransferTaskRequest”对象之后，通过调用createTransferTask的方法创建转储任务。

```
dic.createTransferTask(request);
```

添加转储到数据仓库服务（DWS）的转储任务

```
CreateTransferTaskRequest request = new CreateTransferTaskRequest();

//配置通道名称：用户在数据接入服务(简称DIS)控制台创建通道
request.setStreamName(streamName);

//添加DWS转储任务，并设置任务名称
DwsDestinationDescriptorRequest descriptor = new DwsDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

// 配置DWS集群信息：集群名称、集群ID、数据库等信息。可通过数据仓库服务(简称DWS)控制台创建和查询集群，并通过客户端或其他方式创建数据表
descriptor.setDwsClusterName("dis_test");
descriptor.setDwsClusterId("92f90f6a-de4d-4689-82f6-320c328b0062");
descriptor.setDwsDatabaseName("postgres");
descriptor.setDwsSchema("dbadmin");
descriptor.setDwsTableName("distable01");
descriptor.setDwsDelimiter("|");
descriptor.setUserName(System.getenv("DB_ADMIN"));
descriptor.setUserPassword(System.getenv("DB_PASSWORD"));

//DIS调用KMS服务加密存储DWS的密码，保证用户数据安全：用户可通过数据加密服务(简称KMS)控制台的"密钥管理"创建和查询KMS密钥信息
descriptor.setKmsUserKeyName("qiyinshan");
descriptor.setKmsUserKeyId("9521c600-64a8-4971-ad36-7bbfa6d00c41");

// 转储DWS通过OBS服务中转，需配置OBS桶名和子文件夹名，此目录也用于保存转储失败的源数据文件。可通过OBS控制台或客户端创建桶和文件夹
descriptor.setObsBucketPath("obs-dis");
descriptor.setFilePrefix("transfertask");

// 转储周期，单位s
descriptor.setDeliverTimeInterval(900);

// 可选：在DIS管理页面自动创建dis_admin_agency委托后，默认采用此委托。如未创建过IAM委托，请用主账户登录DIS控制台并创建通道，点击“添加转储任务”，前往授权。
descriptor.setAgencyName("dis_admin_agency");
```



```
// 设置从DIS通道拉取数据时的初始偏移量: 默认LATEST，从通道内最新上传的记录开始读取; TRIM_HORIZON，从通道内最早的未过期记录开始读取
descriptor.setConsumerStrategy(PartitionCursorTypeEnum.LATEST.name());

request.setDwsDestinationDescriptor(descriptor);
```

配置“CreateTransferTaskRequest”对象之后，通过对客户端调用createTransferTask的方法创建转储任务。

```
dic.createTransferTask(request);
```

6.1.6 更新转储任务

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK更新转储任务，需要指定通道名称、转储任务名称，转储周期，转储目标服务等信息。

```
//需配置转储任务的全量参数，不支持更新单个参数
UpdateTransferTaskRequest request = new UpdateTransferTaskRequest();

// 配置待更新的转储任务所属通道的名称
request.setStreamName(streamName);

// 配置待更新的转储任务名称
OBSDestinationDescriptorRequest descriptor = new OBSDestinationDescriptorRequest();
descriptor.setTransferTaskName(taskName);

// 转储至对象存储服务(简称OBS)：OBS桶名和子文件夹名，通过OBS控制台或客户端创建桶和文件夹
descriptor.setObsBucketPath("obs-dis1");
descriptor.setFilePrefix("transfertask");

// 转储周期，单位s
descriptor.setDeliverTimeInterval(300);

// 可选，转储OBS的目标文件格式：默认text，可配置parquet、carbon
descriptor.setDestinationFileType(DestinationFileTypeEnum.TEXT.getType());

request.setObsDestinationDescriptor(descriptor);
```

配置“UpdateTransferTaskRequest”对象之后，通过调用updateTransferTask的方法更新转储任务。

```
dic.updateTransferTask(request);
```

6.1.7 删除转储任务

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK删除指定的转储任务。

```
DeleteTransferTaskRequest request = new DeleteTransferTaskRequest();

// 配置转储任务所属的通道名称
request.setStreamName(streamName);

// 配置待删除的转储任务名称
request.setTransferTaskName(taskName);
```

配置“DeleteTransferTaskRequest”对象之后，通过调用deleteTransferTask的方法删除转储任务。

```
dic.deleteTransferTask(request);
```

6.1.8 查询转储列表

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK查询指定通道的转储任务列表。

```
ListTransferTasksRquest request = new ListTransferTasksRquest();

// 指定待查询的通道名称
request.setStreamName(streamName);
```

配置“ListTransferTaskRequest”对象之后，通过调用listTransferTask的方法查询指定通道的转储任务列表。

```
ListTransferTasksResult result = dic.listTransferTasks(request);
```

查询转储任务列表的返回信息如下。

```
{
   "tasks":[
      {
         "destination_type":"DLI",
         "task_name":"task_Ztab",
         "create_time":1552457808502,
         "state":"RUNNING",
         "last_transfer_timestamp":1552458085454
      },
      {
         "destination_type":"OBS",
         "task_name":"task_qTd9",
         "create_time":1552355757885,
         "state":"RUNNING",
         "last_transfer_timestamp":1552458158527
      }
   ],
   "total_number":2
}
```

6.1.9 查询转储详情

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK查询指定转储任务的详情。

```
DescribeTransferTaskRequest request = new DescribeTransferTaskRequest();

// 指定待查询的通道名称
request.setStreamName(streamName);

// 指定待查询的转储任务名称
request.setTransferTaskName(taskName);
```

配置“DescribeTransferTaskRequest”对象之后，通过调用describeTransferTask的方法查询指定的转储任务详情。

```
DescribeTransferTaskResult result = dic.describeTransferTask(request);
```

查询转储任务的返回信息如下。

```
{
   "partitions":[
      {
         "partitionId":"shardId-0000000000",
         "discard":0,
         "state":"RUNNING",
         "last_transfer_timestamp":1552458085454,
         "last_transfer_offset":56
```

```
        }
    ],
    "stream_name":"dis_test1",
    "task_name":"task_Ztab",
    "task_id":"gGGu2WN88XbmRTm64nJ",
    "destination_type":"DLI",
    "state":"RUNNING",
    "create_time":1552457808502,
    "last_transfer_timestamp":1552458085454,
    "dli_destination_description":{
        "agency_name":"dis_admin_agency",
        "file_prefix":"dli",
        "obs_bucket_path":"dis.test.not.delete",
        "deliver_time_interval":300,
        "consumer_strategy":"LATEST"
    }
}
```

6.1.10 删除通道

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK删除指定的DIS通道。

```
//待删除的通道名称
String streamName = "myStream";
DeleteStreamRequest deleteStreamRequest = new DeleteStreamRequest();
deleteStreamRequest.setStreamName(streamName);
```

配置“DeleteStreamRequest”对象之后，通过对客户端调用deleteStream的方法删除通道。

```
dic.deleteStream(deleteStreamRequest);
```

6.1.11 查询通道列表

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK列出当前活动的通道。

使用setLimit方法设定每次查询时返回的通道数量，若不指定则默认返回的通道数量上限为10。即通道数量少于等于10时显示实际通道数量，通道数量大于10时显示为10。

```
ListStreamsRequest listStreamsRequest = new ListStreamsRequest();
listStreamsRequest.setLimit(5);
System.out.println("listStream: " + JsonUtils.objToJson(dic.listStreams(listStreamsRequest)));
```

表 6-1 请求参数说明

参数名	类型	说明
limit	long	单次请求返回通道列表的最大数量。 取值范围：1~100。 默认值：10。

参数名	类型	说明
exclusiveStartStreamName	string	从该通道开始返回通道列表，返回的通道列表不包括此通道名称。 如果需要分页查询，第一页查询时不传该字段。返回结果has_more_streams为true时，进行下一页查询，exclusiveStartStreamName传入第一页查询结果的最后一条通道名称。

说明

该demo中start_Stream_Name定义为stream0之前的一个通道名称，limit限制为5，所以返回如下信息。

```
listStream: {"total_number":20,"stream_names":
["Stream0","Stream1","Stream2","Stream3","Stream4"],"has_more_streams":true}
```

表 6-2 响应参数说明

参数名	类型	说明
total_number	Int	当前租户所有通道数量。
stream_names	List<String>	满足当前请求条件的通道名称的列表。
has_more_streams	Boolean	是否还有更多满足条件的通道。 • 是：true。 • 否：false。

6.1.12 查询通道详情

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK查询指定通道信息。

```
String streamName = "myStream";
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName(streamName);
System.out.println("descStream: " + JsonUtils.objToJson(dic.describeStream(describeStreamRequest)));
```

查询通道的返回信息如下。

```
descStream: DescribeStreamResult [streamId=JnYbsMfWNn81e8n2mOC, streamName=myStream,
createTime=1540977519187, lastModifiedTime=1540977519983, retentionPeriod=24, status=RUNNING,
streamType=ADVANCED, dataType=BLOB, writablePartitionCount=2, readablePartitionCount=2,
partitions=[PartitionResult{partitionId='shardId-0000000000', hashRange='[0 : 461168601842738790 2]',
status='ACTIVE', parentPartitionIds='null', sequenceNumberRange='[0 : 13286444]'},
PartitionResult{partitionId='shardId-0000000001', hashRange='[4611686018427387903 :
9223372036854775807]', status='ACTIVE', parentPartitionIds='null', sequenceNumberRange='[0 :
13288589]'}], hasMorePartitions=false, updatePartitionCounts=null]
```

6.1.13 下载流式数据

背景信息

下载流式数据，需要确定从分区的什么位置开始获取（即获取游标）。确定起始位置后，再循环获取数据。

获取游标有如下五种方式：

- AT_SEQUENCE_NUMBER
- AFTER_SEQUENCE_NUMBER
- TRIM_HORIZON
- LATEST
- AT_TIMESTAMP

为更好理解游标类型，您需要了解如下几个基本概念。

- 序列号（sequenceNumber）,每个记录的唯一标识符。序列号由DIS在数据生产者调用PutRecord操作以添加数据到DIS数据通道时DIS服务自动分配的。同一分区键的序列号通常会随时间变化增加。PutRecords请求之间的时间段越长，序列号越大。
- 每个分区的sequenceNumber从0开始持续增长，每条数据对应唯一的sequenceNumber，超过生命周期后此sequenceNumber将过期不可用。(例如上传一条数据到新分区，其sequenceNumber起始为0，上传100条之后，则最后一条的sequenceNumber为99；如超过生命周期之后，0~99的数据则不可用)
- 分区的数据有效范围可以通过调用describeStream(查询通道详情)接口获取，其sequenceNumberRange代表数据有效范围，第一个值为最老数据的sequenceNumber，最后一个值为下一条上传数据的sequenceNumber(最新数据的sequenceNumber为此值-1)

  例如[100, 200]，表示此分区总共上传了200条数据，其中第0~99条已过期，有效的最老数据为100，最新数据为199，下一条上传数据的sequenceNumber为200。

场景说明

下表介绍5种下载数据方式的适用场景，您可依据自己的需求进行适配。

表 6-3 场景说明

游标类型（CursorType）	说明	适用场景	备注
AT_SEQUENCE_NUMBER	从特定序列号（即demo中starting-sequence-number定义的序列号）所在的记录开始读取数据。此类型为默认游标类型。	适用于有明确的起始sequenceNumber场景，例如已知需要从哪一条数据开始消费。	与序列号（sequenceNumber）和分区数据有效范围sequenceNumberRange【A，B】强相关。 指定的sequenceNumber应满足如下条件： A<=sequenceNumber<=B
AFTER_SEQUENCE_NUMBER	从特定序列号（即demo中starting-sequence-number定义的序列号）后的记录开始读取数据。	适用于保存了上次消费位置的场景，例如每次消费都保存位置(记录到文件或checkpoint)，若程序重启则可以从保存的位置之后开始恢复，此时用AT_SEQUENCE_NUMBER则会重复一条数据。	与序列号（sequenceNumber）和分区数据有效范围sequenceNumberRange【A，B】强相关。 指定的sequenceNumber应满足如下条件： (A-1)<=sequenceNumber<=(B-1)
TRIM_HORIZON	从分区最老的数据开始消费，即读取分区内所有有效数据。 例如分区数据有效范围为[100, 200]，则会从100开始消费。	适用于不知道消费位置，则直接消费分区内所有有效数据的场景。	无

游标类型（CursorType）	说明	适用场景	备注
LATEST	从分区最新的数据之后开始消费，即不读取分区内的已有数据，而是从下一条上传的数据开始。 (如分区数据有效范围为[100, 200], 则会从200开始消费，如此时无数据上传，则获取的数据为空；如有数据上传，就会得到200,201,202,...)	适用于不知道消费位置，则丢弃分区已有的数据，从新上传的数据开始消费的场景。	无
AT_TIMESTAMP	指定一个时间戳，会从此时间戳上传的数据开始读取，要求在获取游标的时候，有一个明确的13位时间戳。 (例如指定1541742263206，表示读取从2018-11-09 13:44:23开始上传的数据)	适用于不知道消费位置，但想从指定的时间或者从已知上次消费的停止时间开始消费的场景	• 若最老一条数据的上传时间为C，则timestamp>=c即可 • 若timestamp大于最新一条数据的时间戳或者是未来时间，则从最新一条数据之后开始读取。

样例代码

使用**初始化DIS客户端**初始化后的客户端实例通过DIS通道获取数据。

- 下载数据方式选择AT_SEQUENCE_NUMBER和AFTER_SEQUENCE_NUMBER时，样例代码示例如下：

```
//初始化DIS客户端实例
DIS dic = DISClientBuilder.standard()
        .withEndpoint("xxxx")
        // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
        // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
        .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
        .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
        .withProjectId("xxxx")
        .withRegion("xxxx")
        .build();
// 配置通道名称
String streamName = "streamName";
// 配置数据下载分区ID
String partitionId = "shardId-0000000000";
// 配置下载数据序列号
String startingSequenceNumber = "0";
```

```
// 配置下载数据方式
// AT_SEQUENCE_NUMBER: 从指定的sequenceNumber开始获取，需要设置StartingSequenceNumber
// AFTER_SEQUENCE_NUMBER: 从指定的sequenceNumber之后开始获取，需要设置StartingSequenceNumber
String cursorType = PartitionCursorTypeEnum.AT_SEQUENCE_NUMBER.name();

try
{
    // 获取数据游标
    GetPartitionCursorRequest request = new GetPartitionCursorRequest();
    request.setStreamName(streamName);
    request.setPartitionId(partitionId);
    request.setCursorType(cursorType);
    request.setStartingSequenceNumber(startingSequenceNumber);
    GetPartitionCursorResult response = dic.getPartitionCursor(request);
    String cursor = response.getPartitionCursor();
    LOGGER.info("Get stream {}[partitionId={}] cursor success : {}", streamName, partitionId, cursor);
    GetRecordsRequest recordsRequest = new GetRecordsRequest();
    GetRecordsResult recordResponse = null;
    while (true)
        {
            recordsRequest.setPartitionCursor(cursor);
            recordResponse = dic.getRecords(recordsRequest);
            // 下一批数据游标
            cursor = recordResponse.getNextPartitionCursor();

            for (Record record : recordResponse.getRecords())
            {
                LOGGER.info("Get record [{}], partitionKey [{}], sequenceNumber [{}].",
                    new String(record.getData().array()),
                    record.getPartitionKey(),
                    record.getSequenceNumber());
            }


        }
    }
  catch (DISClientException e)
    {
        LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
            e.getMessage(),
            e);
    }
    catch (Exception e)
    {
        LOGGER.error(e.getMessage(), e);
    }
  }
}
```

- 下载数据方式选择TRIM_HORIZON和 LATEST时，样例代码示例如下，参见加粗代码行，基于demo注释掉startingSequenceNumber字段。

```
//初始化DIS客户端实例
DIS dic = DISClientBuilder.standard()
          .withEndpoint("xxxx")
          .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
          .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
          .withProjectId("xxxx")
          .withRegion("xxxx")
          .build();
// 配置通道名称
String streamName = "streamName";
// 配置数据下载分区ID
String partitionId = "shardId-0000000000";
// 配置下载数据序列号
//String startingSequenceNumber = "0";
// 配置下载数据方式
// TRIM_HORIZON: 从最早被存储至分区的有效记录开始读取。
//LATEST：从分区中的最新记录开始读取，此设置可以保证你总是读到分区中最新记录。
String cursorType = PartitionCursorTypeEnum.TRIM_HORIZON.name();
```



```
try
{
    // 获取数据游标
    GetPartitionCursorRequest request = new GetPartitionCursorRequest();
    request.setStreamName(streamName);
    request.setPartitionId(partitionId);
    request.setCursorType(cursorType);
    //request.setStartingSequenceNumber(startingSequenceNumber);
    GetPartitionCursorResult response = dic.getPartitionCursor(request);
    String cursor = response.getPartitionCursor();
    LOGGER.info("Get stream {}[partitionId={}] cursor success : {}", streamName, partitionId, cursor);
    GetRecordsRequest recordsRequest = new GetRecordsRequest();
    GetRecordsResult recordResponse = null;
    while (true)
    {
            recordsRequest.setPartitionCursor(cursor);
            recordsRequest.setLimit(limit);
            recordResponse = dic.getRecords(recordsRequest);
            // 下一批数据游标
            cursor = recordResponse.getNextPartitionCursor();

            for (Record record : recordResponse.getRecords())
            {
                LOGGER.info("Get record [{}], partitionKey [{}], sequenceNumber [{}].",
                    new String(record.getData().array()),
                    record.getPartitionKey(),
                    record.getSequenceNumber());
            }

            if (recordResponse.getRecords().size() == 0)
            {
                Thread.sleep(1000);
            }
        }
    }
    catch (DISClientException e)
    {
        LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
            e.getMessage(),
            e);
    }
    catch (Exception e)
    {
        LOGGER.error(e.getMessage(), e);
    }
  }
}
```

- 下载数据方式选择TAT_TIMESTAMP时，样例代码示例如下，基于demo增加timestamp字段，添加如下加粗行代码。

```
//初始化DIS客户端实例，其中，“endpoint”，“ak”，“sk”，“region”，“projectId”信息请参见。
DIS dic = DISClientBuilder.standard()
          .withEndpoint("xxxx")
          .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
          .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
          .withProjectId("xxxx")
          .withRegion("xxxx")
          .build();
// 配置通道名称
String streamName = "streamName";
// 配置数据下载分区ID
String partitionId = "shardId-0000000000";
// 配置下载数据序列号
//String startingSequenceNumber = "0";
//配置时间戳
long timestamp = 1542960693804L;
// 配置下载数据方式
// AT_TIMESTAMP: 从特定时间戳（即timestamp定义的时间戳）开始读取。
```

```
String cursorType = PartitionCursorTypeEnum.AT_TIMESTAMP.name();

try
{
    // 获取数据游标
    GetPartitionCursorRequest request = new GetPartitionCursorRequest();
    request.setStreamName(streamName);
    request.setPartitionId(partitionId);
    request.setCursorType(cursorType);
    //request.setStartingSequenceNumber(startingSequenceNumber);
    request.setTimestamp(timestamp);
    GetPartitionCursorResult response = dic.getPartitionCursor(request);
    String cursor = response.getPartitionCursor();
    LOGGER.info("Get stream {}[partitionId={}] cursor success : {}", streamName, partitionId, cursor);

        GetRecordsRequest recordsRequest = new GetRecordsRequest();
        GetRecordsResult recordResponse = null;
      while (true)
        {
            recordsRequest.setPartitionCursor(cursor);
            recordsRequest.setLimit(limit);
            recordResponse = dic.getRecords(recordsRequest);
            // 下一批数据游标
            cursor = recordResponse.getNextPartitionCursor();

            for (Record record : recordResponse.getRecords())
            {
                LOGGER.info("Get record [{}], partitionKey [{}], sequenceNumber [{}].",
                    new String(record.getData().array()),
                    record.getPartitionKey(),
                    record.getSequenceNumber());
            }

            if (recordResponse.getRecords().size() == 0)
            {
                Thread.sleep(1000);
            }
        }
    }
    catch (DISClientException e)
    {
        LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
            e.getMessage(),
            e);
    }
    catch (Exception e)
    {
        LOGGER.error(e.getMessage(), e);
    }
  }
}
```

参数说明

表 6-4 参数说明

参数名	参数类型	说明
partitionId	String	分区ID。 **说明** 请根据**上传流式数据**的执行结果，控制台的返回信息字段，例如 “partitionId [shardId-0000000000]”进行定义。

参数名	参数类型	说明
startingSequenceNumber	String	序列号。序列号是每个记录的唯一标识符。序列号由DIS在数据生产者调用PutRecords操作以添加数据到DIS数据通道时DIS服务自动分配的。同一分区键的序列号通常会随时间变化增加。PutRecords请求之间的时间段越长，序列号越大。 **说明** 请根据**上传流式数据**的执行结果，控制台的返回信息字段，例如“sequenceNumber [1]”进行定义。
cursorType	String	游标类型。 • AT_SEQUENCE_NUMBER：从特定序列号（即startingSequenceNumber定义的序列号）所在的记录开始读取数据。此类型为默认游标类型。 • AFTER_SEQUENCE_NUMBER：从特定序列号（即startingSequenceNumber定义的序列号）后的记录开始读取数据。 • TRIM_HORIZON：从最早被存储至分区的有效记录开始读取。 例如，某租户使用DIS的通道，分别上传了三条数据A1，A2，A3。N天后（设定A1已过期，A2和A3仍在有效期范围内），该租户需要下载数据，并选择了TRIM_HORIZON这种下载方式。那么用户可下载的数据将从A2开始读取。 • LATEST：从分区中的最新记录开始读取，此设置可以保证你总是读到分区中最新记录。 • AT_TIMESTAMP：从特定时间戳（即timestamp定义的时间戳）开始读取。

运行程序

右键选择“Run As > 1 Java Application”运行程序，若程序运行成功，可以在控制台查看到类似如下信息：

```
14:55:42.954 [main] INFOcom.bigdata.dis.sdk.DISConfig - get from classLoader
14:55:44.103 [main] INFOcom.bigdata.dis.sdk.util.config.ConfigurationUtils - get from classLoader
14:55:44.105 [main] INFOcom.bigdata.dis.sdk.util.config.ConfigurationUtils - propertyMapFromFile size : 2
14:55:45.235 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get stream
streamName[partitionId=0] cursor success :
eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzLTEzbW9uZXkiLCJwYXJ0aXRpb24taWQiOiIwIiwiY
3Vyc29yLXR5cGUiOiJBVF9TRVFVRU5DRV9OVU1CRVIiLCJzdGFydGluZy1zZXF1ZW5jZS1udW1iZXIiOiIxMDY4O
TcyIn0sImdlbmVyYXRlVGltZXN0YW1wIjoxNTEzNjY2NjMxMTYxfQ
14:55:45.305 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get Record [hello world.],
partitionKey [964885], sequenceNumber [0].
14:55:45.305 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get Record [hello world.],
partitionKey [910960], sequenceNumber [1].
14:55:46.359 [main] INFOcom.bigdata.dis.sdk.demo.ConsumerDemo - Get Record [hello world.],
partitionKey [528377], sequenceNumber [2].
```

表 6-5 参数说明

参数名	参数类型	说明
partition_key	String	用户上传数据时设置的partition_key。 **说明** 上传数据时，如果传了partition_key参数，则下载数据时可返回此参数。如果上传数据时，未传partition_key参数，而是传入partition_id，则不返回partition_key。
startingSequenceNumber	String	序列号。序列号是每个记录的唯一标识符。序列号由DIS在数据生产者调用PutRecords操作以添加数据到DIS数据通道时DIS服务自动分配的。同一分区键的序列号通常会随时间变化增加。PutRecords请求之间的时间段越长，序列号越大。

6.1.14 上传流式数据

样例代码

使用**初始化DIS客户端**后的客户端实例将用户的流式数据通过DIS通道上传至DIS服务。

批量上传流式数据的主体代码如下：

```
//初始化DIS客户端实例，其中，“endpoint”，“ak”，“sk”，“region”，“projectId”信息请参见。
 DIS dic = DISClientBuilder.standard()
          .withEndpoint("xxxx")
          // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
          // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
          .withAk(System.getenv("HUAWEICLOUD_SDK_AK"))
          .withSk(System.getenv("HUAWEICLOUD_SDK_SK"))
          .withProjectId("xxxx")
          .withRegion("xxxx")
          .build();

// 配置通道名称
String streamName = "xxxx";
// 配置上传的数据
String message = "hello world.";

      PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
      putRecordsRequest.setStreamName(streamName);
      List<PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
      // 上传10条
      for (int i = 0; i < 10; i++)
      {
          PutRecordsRequestEntry putRecordsRequestEntry = new PutRecordsRequestEntry();
          putRecordsRequestEntry.setData(ByteBuffer.wrap((message + i).getBytes()));
          putRecordsRequestEntryList.add(putRecordsRequestEntry);
      }
      putRecordsRequest.setRecords(putRecordsRequestEntryList);

      PutRecordsResult putRecordsResult = null;
      try
      {
          putRecordsResult = dic.putRecords(putRecordsRequest);
      }
      catch (DISClientException e)
      {
          LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
```

```
            e.getMessage(),
            e);
        }
        catch (Exception e)
        {
            LOGGER.error(e.getMessage(), e);
        }
// 用户业务需要对上传结果逐条进行判断，确认是否上传成功
        if (putRecordsResult != null)
        {
            LOGGER.info("Put {} [{} successful / {} failed] records.",
                putRecordsResult.getRecords().size(),
                putRecordsResult.getRecords().size() - putRecordsResult.getFailedRecordCount().get(),
                putRecordsResult.getFailedRecordCount());

            for (int j = 0; j < putRecordsResult.getRecords().size(); j++)
            {
                PutRecordsResultEntry putRecordsRequestEntry = putRecordsResult.getRecords().get(j);
                if (!StringUtils.isNullOrEmpty(putRecordsRequestEntry.getErrorCode()))
                {
                    // 上传失败
                    LOGGER.error("[{}] put failed, errorCode [{}], errorMessage [{}]",
                        new String(putRecordsRequestEntryList.get(j).getData().array()),
                        putRecordsRequestEntry.getErrorCode(),
                        putRecordsRequestEntry.getErrorMessage());
                }
                else
                {
                    // 上传成功
                    LOGGER.info("[{}] put success, partitionId [{}], sequenceNumber [{}]",
                        new String(putRecordsRequestEntryList.get(j).getData().array()),
                        putRecordsRequestEntry.getPartitionId(),
                        putRecordsRequestEntry.getSequenceNumber());
                }
            }
        }
```

逐条上传流式数据的主体代码如下：

```
// 创建DIS客户端实例
DIS dic = DISUtil.getInstance();

// 配置流名称
String streamName = DISUtil.getStreamName();

// 配置上传的数据
String message = "Hello world";

        PutRecordRequest putRecordsRequest = new PutRecordRequest();
        putRecordsRequest.setStreamName(streamName);
        putRecordsRequest.setData(ByteBuffer.wrap((message).getBytes()));
        try
        {
            PutRecordResult putRecordsResult = dic.putRecord(putRecordsRequest);
            // 上传成功
            LOGGER.info("[{}] put success, partitionId [{}], sequenceNumber [{}]",
                message, putRecordsResult.getPartitionId(), putRecordsResult.getSequenceNumber());
        }
        catch (DISClientException e)
        {
            LOGGER.error("Failed to get a normal response, please check params and retry. Error message [{}]",
                e.getMessage(), e);
        }
        catch (Exception e)
        {
            LOGGER.error(e.getMessage(), e);
        }
```


运行程序

右键选择“Run As > 1 Java Application”运行程序，若程序运行成功，可以在控制台查看到类似如下信息：

```
15:19:29.298 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - ========== BEGIN PUT
============
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - Put 3 records[3 successful / 0 failed].
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - [hello world.] put success, partitionId
[shardId-0000000000], sequenceNumber [1]
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - [hello world.] put success, partitionId
[shardId-0000000000], sequenceNumber [2]
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - [hello world.] put success, partitionId
[shardId-0000000000], sequenceNumber [3]
15:19:30.992 [main] INFO  com.bigdata.dis.sdk.demo.ProducerDemo - ========== END PUT ============
```

6.1.15 获取数据游标

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK获取数据游标信息。

```
// 配置通道名称
String streamName = "myStream";
// 配置数据下载分区ID
String partitionId = "0";
// 配置下载数据序列号
String startingSequenceNumber = "0";
// 配置下载数据方式
String cursorType = PartitionCursorTypeEnum.AT_SEQUENCE_NUMBER.name();

GetPartitionCursorRequest request = new GetPartitionCursorRequest();
request.setStreamName(streamName);
request.setPartitionId(partitionId);
request.setStartingSequenceNumber(startingSequenceNumber);
request.setCursorType(cursorType);
GetPartitionCursorResult response = dic.getPartitionCursor(request);
String cursor = response.getPartitionCursor();
```

6.1.16 创建 APP

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK创建APP，需要指定APP名称。

```
// APP名称
String appName = "myApp";
```

配置APP名称之后，通过调用createApp的方法创建APP。

```
dic.createApp(myApp);
```

6.1.17 删除 APP

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK创建DIS通道，需要指定APP名称。

```
// 待删除APP名称
String appName = "myApp";
```

配置APP名称之后，通过调用deleteApp的方法删除APP。

```
dic.deleteApp(myApp);
```

6.1.18 新增 Checkpoint

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

使用DIS SDK创建Checkpoint，需要指定通道名称、APP名称、分区编号、序列号以及Checkpoint类型。

```
// 通道名称
String streamName = "myStream";
// APP名称
String appName = "myApp";
CommitCheckpointRequest commitCheckpointRequest = new CommitCheckpointRequest();
commitCheckpointRequest.setStreamName(streamName);
commitCheckpointRequest.setAppName(appName);
// 需要提交的sequenceNumber
commitCheckpointRequest.setSequenceNumber("100");
// 分区编号
commitCheckpointRequest.setPartitionId("0");
// Checkpoint类型
commitCheckpointRequest.setCheckpointType(CheckpointTypeEnum.LAST_READ.name());
```

配置“CommitCheckpointRequest”对象之后，通过调用 commitCheckpoint 提交checkpoint。

```
dic.commitCheckpoint(commitCheckpointRequest);
```

6.1.19 查询 Checkpoint

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

```
GetCheckpointRequest getCheckpointRequest = new GetCheckpointRequest();
// 设置App名称(注: 1.3.0及以前版本使用setAppId，1.3.1及后续版本请使用setAppName
getCheckpointRequest.setAppName(appName);
getCheckpointRequest.setStreamName(streamName);
// 分区编号
getCheckpointRequest.setPartitionId("0");
// Checkpoint类型
getCheckpointRequest.setCheckpointType(CheckpointTypeEnum.LAST_READ.name());
System.out.println("getCheckpoint: " + JsonUtils.objToJson(dic.getCheckpoint(getCheckpointRequest)));
```

查询Checkpoint的返回信息如下。

```
getCheckpoint:
{"sequence_number": "10", "metadata": "metadata"}
```

6.1.20 变更分区数量

参考**初始化DIS客户端**的操作初始化一个DIS客户端实例，实例名称为dic。

```
// 目标分区数量
int targetPartitionCount = 2;

UpdatePartitionCountRequest update = new UpdatePartitionCountRequest();
update.setStreamName(streamName);
update.setTargetPartitionCount(targetPartitionCount);
try
{
    UpdatePartitionCountResult updatePartitionCountResult = dic.updatePartitionCount(update);
    LOGGER.info("Success to update partition count, {}", updatePartitionCountResult);
}
catch (Exception e)
{
    LOGGER.error("Failed to update partition count", e);
}
```

变更分区数量成功的返回信息如下。

```
Success to update partition count, UpdatePartitionCountResult [currentPartitionCount=2,
streamName=mystream, targetPartitionCount=2]
```

6.2 使用 Kafka Adapter 上传与下载数据

6.2.1 Kafka Adapter 概述

dis-kafka-adapter是数据接入服务（DIS）提供的一个sdk，支持原本使用Kafka Client上传数据的用户，以类似原来的操作将数据上传到DIS。

目前只支持Java版本。

6.2.2 准备环境

配置 pom.xml 文件

如果已有maven工程，在pom.xml中使用如下依赖即可。

```
<dependency>
    <groupId>com.huaweicloud.dis</groupId>
    <artifactId>huaweicloud-dis-kafka-adapter</artifactId>
    <version>1.2.18</version>
</dependency>
```

使用 DIS 样例工程

在**https://dis-publish.obs-website.cn-north-1.myhuaweicloud.com/**中下载DIS的kafka-adapter压缩包。

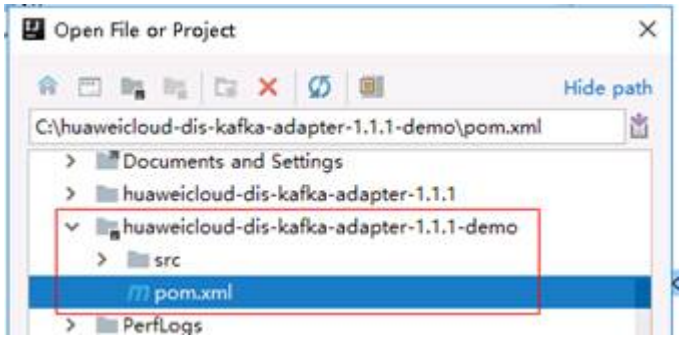
此zip包中有两个目录。

- huaweicloud-dis-kafka-adapter-*X.X.X*目录下是所有依赖的jar包，如果使用非Maven工程，则可导入此lib目录下的所有jar到环境依赖即可
- huaweicloud-dis-kafka-adapter-*X.X.X*-demo是一个样例工程，使用maven编写

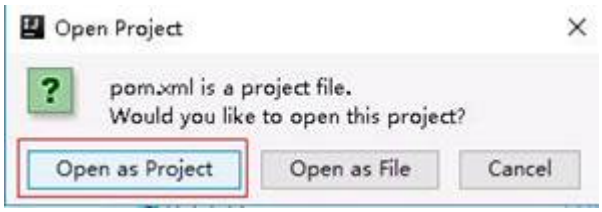
例如使用IntelliJ IDEA，可按如下方式导入工程。

步骤1 打开IntelliJ IDEA，选择“File > Open”

在弹出的对话框中，选择huaweicloud-dis-kafka-adapter-*X.X.X*-demo目录，并双击pom.xml确定。

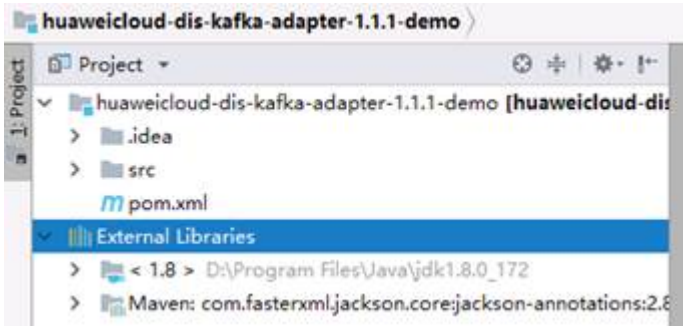


步骤2 当弹出如下对话框，请选择“Open as Project”，作为工程打开。



步骤3 单击“New Window”，在新窗口打开此工程。

步骤4 等待IntelliJ IDEA构建工程，完成之后会显示目录与文件



----结束

检查认证信息

- 检查AK/SK

  AK/SK (Access Key ID/Secret Access Key)是用户调用接口的访问密钥。

  您可以通过如下方式获取访问密钥。

  a. 登录控制台，在用户名下拉列表中选择“我的凭证”。

  b. 进入“我的凭证”页面，选择“访问密钥 > 新增访问密钥”，如图6-3所示。

  图 6-3 单击新增访问密钥

  c. 单击“确定”，根据浏览器提示，保存密钥文件。密钥文件会直接保存到浏览器默认的下载文件夹中。打开名称为“credentials.csv”的文件，即可查看访问密钥（Access Key Id和Secret Access Key）。

说明

- 每个用户仅允许新增两个访问密钥。
- 为保证访问密钥的安全，访问密钥仅在初次生成时自动下载，后续不可再次通过管理控制台界面获取。请在生成后妥善保管。

- 检查项目ID

  项目ID表示租户的资源，账号ID对应当前账号，IAM用户ID对应当前用户。用户可在对应页面下查看不同Region对应的项目ID、账号ID和用户ID。

  a. 注册并登录管理控制台。
  b. 在用户名的下拉列表中单击“我的凭证”。
  c. 在“API凭证”页面，查看账号名和账号ID、IAM用户名和IAM用户ID，在项目列表中查看项目和项目ID。

6.2.3 上传数据

代码样例

“ak”、“sk”和“projectId”信息的获取请参见**检查认证信息**。

```
package com.huaweicloud.dis.demo.adapter;
import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.producer.*;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Future;
import java.util.concurrent.ThreadLocalRandom;

public class DISKafkaProducerDemo
{
    private static final Logger LOGGER = LoggerFactory.getLogger(DISKafkaProducerDemo.class);


    public static void main(String[] args)
    {

        // 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
        // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK    。
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
        String sk = System.getenv("HUAWEICLOUD_SDK_SK");
        // YOU ProjectId
        String projectId = "YOU_PROJECT_ID";
       // YOU DIS Stream
        String streamName = "YOU_STREAM_NAME";
       // 消费组ID，用于记录offset
       String groupId = "YOU_GROUP_ID";
      // DIS region
       String region = "your region";

        Properties props = new Properties();
        props.setProperty(DISConfig.PROPERTY_AK, ak);
        props.setProperty(DISConfig.PROPERTY_SK, sk);
        props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
        props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
        props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
```

```
        // 默认情况下不需要设置endpoint，会自动使用域名访问；如需使用指定的endpoint，解除如下注释并设置endpoint即可
        // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

        // Create dis producer
        Producer<String, String> producer = new DISKafkaProducer<>(props);

        // 同步发送
        synchronousSendDemo(producer, streamName);

        // 异步发送
        asynchronousSendDemo(producer, streamName);

        // 关闭producer，防止资源泄露
        producer.close();
    }

    public static void synchronousSendDemo(Producer<String, String> producer, String streamName)
    {
        LOGGER.info("===== synchronous send =====");
        for (int i = 0; i < 5; i++)
        {
            // key设置为随机(或者设置为null)，数据会均匀分配到所有分区中
            String key = String.valueOf(ThreadLocalRandom.current().nextInt(1000000));
            String value = "Hello world[sync]. " + i;

            Future<RecordMetadata> future = producer.send(new ProducerRecord<>(streamName, key, value));

            try
            {
                // 调用future.get会阻塞等待，直到发送完成
                RecordMetadata recordMetadata = future.get();
                // 发送成功
                LOGGER.info("Success to send [{}], Partition [{}], Offset [{}].",
                        value, recordMetadata.partition(), recordMetadata.offset());
            }
            catch (Exception e)
            {
                // 发送失败
                LOGGER.error("Failed to send [{}], Error [{}]", value, e.getMessage(), e);
            }
        }
    }

    public static void asynchronousSendDemo(Producer<String, String> producer, String streamName)
    {
        LOGGER.info("===== asynchronous send =====");
        int totalSendCount = 5;
        CountDownLatch countDownLatch = new CountDownLatch(totalSendCount);
        for (int i = 0; i < totalSendCount; i++)
        {
            // key设置为随机(或者设置为null)，数据会均匀分配到所有分区中
            String key = String.valueOf(ThreadLocalRandom.current().nextInt(1000000));
            String value = "Hello world[async]. " + i;

            try
            {
                // 使用回调方式发送，不会阻塞
                producer.send(new ProducerRecord<>(streamName, key, value), new Callback()
                {
                    @Override
                    public void onCompletion(RecordMetadata recordMetadata, Exception e)
                    {
                        countDownLatch.countDown();
                        if (e == null)
                        {
                            // 发送成功
                            LOGGER.info("Success to send [{}], Partition [{}], Offset [{}].",
                                    value, recordMetadata.partition(), recordMetadata.offset());
```

```
                    }
                    else
                    {
                        // 发送失败
                        LOGGER.error("Failed to send [{}], Error [{}]", value, e.getMessage(), e);
                    }
                }
            });
        }
        catch (Exception e)
        {
            countDownLatch.countDown();
            LOGGER.error(e.getMessage(), e);
        }
    }

    try
    {
        // 等待所有发送完成
        countDownLatch.await();
    }
    catch (InterruptedException e)
    {
        LOGGER.error(e.getMessage(), e);
    }
  }
}
```

执行如上程序，发送数据成功会打印如下日志

```
09:32:52.001 INFO  c.h.d.d.a.DISKafkaProducerDemo - ===== synchronous send =====
09:32:53.523 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 0], Partition [0],
Offset [114].
09:32:53.706 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 1], Partition [0],
Offset [115].
09:32:53.956 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 2], Partition [0],
Offset [116].
09:32:54.160 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 3], Partition [0],
Offset [117].
09:32:54.450 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[sync]. 4], Partition [0],
Offset [118].
09:32:54.450 INFO  c.h.d.d.a.DISKafkaProducerDemo - ===== asynchronous send =====
09:32:54.673 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 0], Partition [0],
Offset [119].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 1], Partition [0],
Offset [120].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 2], Partition [0],
Offset [121].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 3], Partition [0],
Offset [122].
09:32:54.674 INFO  c.h.d.d.a.DISKafkaProducerDemo - Success to send [Hello world[async]. 4], Partition [0],
Offset [123].
```

与原生 KafkaProducer 接口适配说明

DISKafkaProducer的实现与KafkaProducer的实现不同，DISKafkaProducer的客户端与服务端通过Rest API实现，而KafkaProducer是基于TCP协议实现，在接口兼容上有如下差异。

表 6-6 适配说明

原生KafkaProducer	类型	DISKafkaProducer	说明
Future<RecordMetadata> send(ProducerRecord<K, V> record)	接口	支持	发送单条数据
Future<RecordMetadata> send(ProducerRecord<K, V> record, Callback callback)	接口	支持	发送单条数据并设置回调处理函数
void close()	接口	支持	关闭Producer
void close(long timeout, TimeUnit timeUnit)	接口	支持	关闭Producer并设置超时时间
List<PartitionInfo> partitionsFor(String topic)	接口	支持	获取通道的分区信息
void flush(long timeout, TimeUnit unit)	接口	不支持	强制发送当前缓存数据，后续支持
Map<MetricName, ? extends Metric> metrics()	接口	不支持	获取统计信息
key.serializer	参数	支持	含义与kafka设置相同，但默认值为StringSerializer (kafka必须配置)
value.serializer	参数	支持	含义与kafka设置相同，但默认值为StringSerializer (kafka必须配置)
linger.ms	参数	支持	含义与kafka设置相同，但默认值为50(kafka是0)，目的是提高Rest接口的上传效率
batch.size	参数	支持	含义与kafka设置相同，但默认值为1MB(kafka是16KB)，目的是匹配流控的大小
buffer.memory	参数	支持	同kafka的默认设置(32MB)
max.in.flight.requests.per.connection	参数	支持	限制客户端在单个连接上能够发送的未响应请求的个数，默认值为100(Kafka默认为5)可提高发送性能，但可能出现数据顺序不一致的问题。如需严格保证顺序，建议此值设置为1

原生KafkaProducer	类型	DISKafkaProducer	说明
block.on.buffer.full	参数	支持	同Kafka默认设置(false)。 ● true表示当发送缓冲区满，send一直阻塞不超时； ● false表示发送缓冲区满后根据max.block.ms的时间阻塞，超过时间则抛出异常。
max.block.ms	参数	支持	同Kafka默认设置(60000)。 当发送缓冲区满且block.on.buffer.full为false时，控制send()的阻塞时间(毫秒)。
retries	参数	支持，但是参数名改为exception.retries	Kafka默认设置为0，DIS默认设置为8。 出现网络/服务端异常的重试次数，尽量保证数据上传成功
其他参数	参数	不支持	-

6.2.4 数据下载的消费模式

同Kafka类似，当前dis kafka adapter支持三种消费模式。

assign 模式

由用户手动指定consumer实例消费哪些具体分区，此时不会拥有group management机制，也就是当group内消费者数量变化或者通道扩缩容的时候不会有重新分配分区的行为发生。代码样例如下所示：

```
package com.huaweicloud.dis.demo.adapter;

import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.consumer.*;
import com.huaweicloud.dis.adapter.kafka.common.PartitionInfo;
import com.huaweicloud.dis.adapter.kafka.common.TopicPartition;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Properties;

public class DISKafkaConsumerAssignDemo
{
    private static final Logger LOGGER = LoggerFactory.getLogger(DISKafkaConsumerAssignDemo.class);

    public static void main(String[] args)
  {

      // 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
      // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量
```

```
HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK    。
      String ak = System.getenv("HUAWEICLOUD_SDK_AK");
      String sk = System.getenv("HUAWEICLOUD_SDK_SK");
      // YOU ProjectId
      String projectId = "YOU_PROJECT_ID";
     // YOU DIS Stream
      String streamName = "YOU_STREAM_NAME";
     // 消费组ID，用于记录offset
     String groupId = "YOU_GROUP_ID";
    // DIS region
     String region = "your region";

       Properties props = new Properties();
       props.setProperty(DISConfig.PROPERTY_AK, ak);
       props.setProperty(DISConfig.PROPERTY_SK, sk);
       props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
       props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
       props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
       props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
       props.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
       props.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
       props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
OffsetResetStrategy.LATEST.name());


      // 默认情况下不需要设置endpoint，会自动使用域名访问；如需使用指定的endpoint，解除如下注释并设
置endpoint即可
      // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

      Consumer<String, String> consumer = new DISKafkaConsumer<>(props);
      List<TopicPartition> topicPartitions = new ArrayList<>();
      for (PartitionInfo partitionInfo : consumer.partitionsFor(streamName))
      {
          topicPartitions.add(new TopicPartition(partitionInfo.topic(), partitionInfo.partition()));
      }

      // 使用assign模式，指定需要消费的分区
      consumer.assign(topicPartitions);

      while (true)
      {
          try
          {
              ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

              if (!records.isEmpty())
              {
                  for (TopicPartition partition : records.partitions())
                  {
                      List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
                      for (ConsumerRecord<String, String> record : partitionRecords)
                      {
                          LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                                  record.value(), record.partition(), record.offset(), record.key());
                      }
                  }
                  // 数据处理完成之后异步提交当前offset(也可使用同步提交commitSync)
                  consumer.commitAsync(new OffsetCommitCallback()
                  {
                      @Override
                      public void onComplete(Map<TopicPartition, OffsetAndMetadata> map, Exception e)
                      {
                          if (e == null)
                          {
                              LOGGER.debug("Success to commit offset [{}]", map);
                          }
                          else
```

```
                    {
                        LOGGER.error("Failed to commit offset [{}]", e.getMessage(), e);
                    }
                }
            });
        }
    }
    catch (Exception e)
    {
        LOGGER.info(e.getMessage(), e);
    }
  }
 }
}
```

执行如上程序之后，如果有数据发送到通道中，此时会打印如下日志。

```
09:36:45.071 INFO  c.h.d.a.k.c.DISKafkaConsumer - create DISKafkaConsumer successfully
09:36:49.842 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 0], Partition [0],
Offset [134], Key [769066]
09:36:49.963 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 1], Partition [0],
Offset [135], Key [700005]
09:36:50.145 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 2], Partition [0],
Offset [136], Key [338044]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 3], Partition [0],
Offset [137], Key [537495]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[sync]. 4], Partition [0],
Offset [138], Key [980016]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 0], Partition [0],
Offset [139], Key [182772]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 1], Partition [0],
Offset [140], Key [830271]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 2], Partition [0],
Offset [141], Key [927054]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 3], Partition [0],
Offset [142], Key [268122]
09:36:51.093 INFO  c.h.d.d.a.DISKafkaConsumerAssignDemo - Value [Hello world[async]. 4], Partition [0],
Offset [143], Key [992787]
```

subscribe 模式

用户指定通道名称即可，无需指定具体分区，服务端会根据消费者的数量变化或者通道扩缩容变化，触发group management机制，自动给每一个消费者分配分区，保证一个分区只会被一个消费者消费。

代码样例如下所示：

```
package com.huaweicloud.dis.demo.adapter;

import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.consumer.*;
import com.huaweicloud.dis.adapter.kafka.common.TopicPartition;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.Properties;

public class DISKafkaConsumerSubscribeDemo
{
    private static final Logger LOGGER = LoggerFactory.getLogger(DISKafkaConsumerSubscribeDemo.class);

    public static void main(String[] args)
```

```
 {

    // 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
    // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK    。
     String ak = System.getenv("HUAWEICLOUD_SDK_AK");
     String sk = System.getenv("HUAWEICLOUD_SDK_SK");
     // YOU ProjectId
     String projectId = "YOU_PROJECT_ID";
    // YOU DIS Stream
     String streamName = "YOU_STREAM_NAME";
    // 消费组ID，用于记录offset
    String groupId = "YOU_GROUP_ID";
   // DIS region
    String region = "your region";

      Properties props = new Properties();
      props.setProperty(DISConfig.PROPERTY_AK, ak);
      props.setProperty(DISConfig.PROPERTY_SK, sk);
      props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
      props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
      props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
      props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
      props.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
      props.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
      props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
OffsetResetStrategy.LATEST.name());

      // 默认情况下不需要设置endpoint，会自动使用域名访问；如需使用指定的endpoint，解除如下注释并设置endpoint即可
      // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

      Consumer<String, String> consumer = new DISKafkaConsumer<>(props);
      // 使用subscribe模式，指定需要消费的通道名即可
      consumer.subscribe(Collections.singleton(streamName));

      while (true)
      {
         try
         {
            ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

            if (!records.isEmpty())
            {
               for (TopicPartition partition : records.partitions())
               {
                  List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
                  for (ConsumerRecord<String, String> record : partitionRecords)
                  {
                     LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                           record.value(), record.partition(), record.offset(), record.key());
                  }
               }
               // 数据处理完成之后异步提交当前offset(也可使用同步提交commitSync)
               consumer.commitAsync(new OffsetCommitCallback()
               {
                  @Override
                  public void onComplete(Map<TopicPartition, OffsetAndMetadata> map, Exception e)
                  {
                     if (e == null)
                     {
                        LOGGER.debug("Success to commit offset [{}]", map);
                     }
                     else
                     {
                        LOGGER.error("Failed to commit offset [{}]", e.getMessage(), e);
```

```
                    }
                }
            });
        }
    }
    catch (Exception e)
    {
        LOGGER.info(e.getMessage(), e);
    }
  }
 }
}
```

程序启动后，会每10s发起心跳请求(Heartbeat)，然后发起加入消费组的请求(JoinGroup)，服务端此时会开始给此消费组中的消费者分配分区，此过程大约需要等待20s，完成之后消费者会发起同步请求(SyncGroup)获取分配结果，等日志中输出Heartbeat {"state":"STABLE"}的信息，表示整个消费组都完成分配，可以正常消费数据了。

此过程的关键日志说明如下

- Heartbeat {"state":"JOINING"}

Heartbeat表示心跳请求，每10s发起一次，用于和服务端保持连接。如果超过1分钟服务端没有收到心跳，会认为消费端已离线，消费组会重新分配。若心跳结果为JOINING表示消费者需要重新加入消费组，若为STABLE表示消费组稳定。

- JoinGroup

如果Heartbeat的结果不为STABLE，则消费者会发起joinGroup的请求，通知服务端自己要加入消费组，服务端收到客户端的join请求之后，会将消费组重新分配，此时返回一个syncDelayedTimeMs，告诉客户端分配需要多久完成，客户端可以等待syncDelayedTimeMs之后，再发起同步请求(SyncGroup)获取分配结果

- SyncGroup

此请求用于获取分配结果，返回的assignment中即为消费者需要消费的通道名和分区

执行样例程序，等待消费组分配完成之后，发送数据到通道，完整的日志如下

```
09:42:37.296 INFO  c.h.d.a.k.c.DISKafkaConsumer - create DISKafkaConsumer successfully
09:42:37.354 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"JOINING"}
09:42:37.363 INFO  c.h.d.a.k.c.Coordinator - joinGroupRequest {"groupId":"ding","clientId":"consumer-c2d43144-0823-4eea-aaa8-7af95c536144","interestedStream":["liuhao12"]}
09:42:37.406 INFO  c.h.d.a.k.c.Coordinator - joinGroupResponse {"state":"OK","syncDelayedTimeMs":21000}
09:42:58.408 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"groupId":"ding","clientId":"consumer-c2d43144-0823-4eea-aaa8-7af95c536144","generation":-1}
09:42:58.451 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"state":"OK","generation":33,"assignment":{"dis-test":[0]}}
09:42:58.488 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
09:43:08.960 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
09:46:56.227 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 0], Partition [0], Offset [144], Key [799704]
09:46:56.327 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 1], Partition [0], Offset [145], Key [683757]
09:46:56.449 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 2], Partition [0], Offset [146], Key [439062]
09:46:56.535 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 3], Partition [0], Offset [147], Key [374939]
09:46:56.654 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[sync]. 4], Partition [0], Offset [148], Key [321528]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 0], Partition [0], Offset [149], Key [964841]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 1], Partition [0], Offset [150], Key [520262]
```

```
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 2], Partition
[0], Offset [151], Key [619119]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 3], Partition
[0], Offset [152], Key [257094]
09:46:56.749 INFO  c.h.d.d.a.DISKafkaConsumerSubscribeDemo - Value [Hello world[async]. 4], Partition
[0], Offset [153], Key [310331]
```

subscribePattern 模式

subscribePattern是在subscribe的基础上，用户不用指定具体的通道名称而是使用通配符，例如stream.* 表示会消费 stream1, stream2, stream_123等等。已有/新增/删除的通道，只要匹配通配符，就可被消费组消费。

代码样例如下所示：

```
package com.huaweicloud.dis.demo.adapter;

import com.huaweicloud.dis.DISConfig;
import com.huaweicloud.dis.adapter.kafka.clients.consumer.*;
import com.huaweicloud.dis.adapter.kafka.common.TopicPartition;
import com.huaweicloud.dis.adapter.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.regex.Pattern;

public class DISKafkaConsumerSubscribePatternDemo
{
    private static final Logger LOGGER =
LoggerFactory.getLogger(DISKafkaConsumerSubscribePatternDemo.class);

    public static void main(String[] args)
    {
        // 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
        // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK    。
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
        String sk = System.getenv("HUAWEICLOUD_SDK_SK");
        // YOU ProjectId
        String projectId = "YOU_PROJECT_ID";
       // YOU DIS Stream
        String streamName = "YOU_STREAM_NAME";
       // 消费组ID，用于记录offset
       String groupId = "YOU_GROUP_ID";
       // DIS region
       String region = "your region";

         Properties props = new Properties();
         props.setProperty(DISConfig.PROPERTY_AK, ak);
         props.setProperty(DISConfig.PROPERTY_SK, sk);
         props.setProperty(DISConfig.PROPERTY_PROJECT_ID, projectId);
         props.setProperty(DISConfig.PROPERTY_REGION_ID, region);
         props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
         props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
         props.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
         props.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
         props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
OffsetResetStrategy.LATEST.name());

         // 默认情况下不需要设置endpoint，会自动使用域名访问；如需使用指定的endpoint，解除如下注释并设置endpoint即可
```

```
    // props.setProperty(DISConfig.PROPERTY_ENDPOINT, "https://dis-${region}.myhuaweicloud.com");

    Consumer<String, String> consumer = new DISKafkaConsumer<>(props);
    // 使用subscribePattern模式，指定通配符即可
    consumer.subscribe(Pattern.compile(streamNamePattern), new ConsumerRebalanceListener()
    {
        @Override
        public void onPartitionsRevoked(Collection<TopicPartition> collection)
        {
            LOGGER.info("onPartitionsRevoked [{}]", collection);
        }

        @Override
        public void onPartitionsAssigned(Collection<TopicPartition> collection)
        {
            LOGGER.info("onPartitionsAssigned [{}]", collection);
        }
    });

    while (true)
    {
        try
        {
            ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

            if (!records.isEmpty())
            {
                for (TopicPartition partition : records.partitions())
                {
                    List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
                    for (ConsumerRecord<String, String> record : partitionRecords)
                    {
                        LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                                record.value(), record.partition(), record.offset(), record.key());
                    }
                }
                // 数据处理完成之后异步提交当前offset(也可使用同步提交commitSync)
                consumer.commitAsync(new OffsetCommitCallback()
                {
                    @Override
                    public void onComplete(Map<TopicPartition, OffsetAndMetadata> map, Exception e)
                    {
                        if (e == null)
                        {
                            LOGGER.debug("Success to commit offset [{}]", map);
                        }
                        else
                        {
                            LOGGER.error("Failed to commit offset [{}]", e.getMessage(), e);
                        }
                    }
                });
            }
        }
        catch (Exception e)
        {
            LOGGER.info(e.getMessage(), e);
        }
    }
  }
}
```

程序启动之后，仍然需要等待约20s，服务端分配完成之后，才可以开始消费数据，样例代码执行的日志如下所示

```
10:10:36.420 INFO  c.h.d.a.k.c.DISKafkaConsumer - create DISKafkaConsumer successfully
10:10:36.481 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"JOINING"}
10:10:36.486 INFO  c.h.d.a.k.c.Coordinator - joinGroupRequest {"groupId":"ding","clientId":"consumer-
cad967ba-70ab-4e02-b184-f60b95fe3256","streamPattern":"stream.*"}
10:10:36.697 INFO  c.h.d.a.k.c.Coordinator - joinGroupResponse {"state":"OK","subscription":
```

```
["stream_hello","stream_world"],"syncDelayedTimeMs":21000}
10:10:57.699 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"groupId":"ding","clientId":"consumer-
cad967ba-70ab-4e02-b184-f60b95fe3256","generation":-1}
10:10:57.746 INFO  c.h.d.a.k.c.Coordinator - syncGroup {"state":"OK","generation":34,"assignment":
{"stream_hello":[0],"stream_world":[0]}}
10:10:57.770 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - onPartitionsAssigned
[[stream_hello-0, stream_world-0]]
10:10:57.770 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
10:11:08.466 INFO  c.h.d.a.k.c.Coordinator - Heartbeat {"state":"STABLE"}
10:11:09.992 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 0],
Partition [0], Offset [154], Key [181881]
10:11:09.993 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 1],
Partition [0], Offset [155], Key [483023]
10:11:09.993 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 2],
Partition [0], Offset [156], Key [32453]
10:11:10.093 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 3],
Partition [0], Offset [157], Key [111948]
10:11:10.180 INFO  c.h.d.d.a.DISKafkaConsumerSubscribePatternDemo - Value [Hello world[sync]. 4],
Partition [0], Offset [158], Key [822860]
```

6.2.5 下载数据之消费位移

消费位移确认有自动提交与手动提交两种策略，在创建DISKafkaConsumer对象时，通过参数enable.auto.commit设定，true表示自动提交（默认）。

自动提交策略由消费者协调器（Coordinator）每隔${auto.commit.interval.ms}毫秒执行一次偏移量的提交；手动提交需要由客户端自己控制偏移量的提交。

- 自动提交

  在创建一个消费者时，默认是自动提交偏移量，默认的提交间隔是5000ms。使用自动提交相关参数设置如下：

```
props.setProperty("enable.auto.commit", "true");// 显示设置偏移量自动提交
props.setProperty("auto.commit.interval.ms", "5000");// 设置偏移量提交时间间隔
```

- 手动提交

在有些场景可能对消费偏移量有更精确的管理，以保证消息不被重复消费以及消息不被丢失。假设对拉取到的消息需要进行写入数据库处理，或者用于其他网络访问请求等等复杂的业务处理，在这种场景下，所有的业务处理完成后才认为消息被成功消费，此时必须手动控制偏移量的提交。使用手动提交相关参数设置如下：

```
props.put("enable.auto.commit", "false");// 显示设置偏移量手动提交
```

然后在业务处理成功后调用commitAsync()或commitSync()方法提交偏移量：commitAsync()是异步提交，消费者线程不会被阻塞，可能在提交偏移量操作的结果还未返回时就开始进行下一次的拉取操作，如需获取提交的结果，可以添加回调方法OffsetCommitCallback，当提交偏移量完成后会自动调用其onComplete()方法，以便根据回调结果执行不同的逻辑处理；

commitSync()是同步提交，会阻塞线程直到提交消费偏移量执行结果返回。

另外还可以精细的控制对具体分区具体offset数据的确认，确认的offset为已接受数据最大offset+1。例如消费一批数据，最后一条的offset为100，则此时需要commit 101，这样下次消费就会从101开始，不会重复。代码样例如下：

```
ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);

if (!records.isEmpty())
{
   for (TopicPartition partition : records.partitions())
   {
      List<ConsumerRecord<String, String>> partitionRecords = records.records(partition);
      for (ConsumerRecord<String, String> record : partitionRecords)
```

```
        {
            LOGGER.info("Value [{}], Partition [{}], Offset [{}], Key [{}]",
                    record.value(), record.partition(), record.offset(), record.key());
        }
        if (!partitionRecords.isEmpty())
        {
            // 同步确认某个分区的特定offset
            long lastOffset = partitionRecords.get(partitionRecords.size() - 1).offset();
            consumer.commitSync(Collections.singletonMap(partition, new OffsetAndMetadata(lastOffset +
1)));
        }
    }
}
```

6.2.6 与原生 KafkaConsumer 接口适配说明

表 6-7 接口适配说明

原生KafkaConsumer	类型	DISKafkaConsumer	说明
Set<TopicPartition> assignment()	接口	支持	获取consumer消费的通道与分区信息
Set<String> subscription()	接口	支持	获取consumer已订阅的通道名称
void assign(Collection<TopicPartition> var1)	接口	支持	分配指定的分区
void subscribe(Collection<String> var1)	接口	支持	订阅指定的通道
void subscribe(Collection<String> var1, ConsumerRebalanceListener var2)	接口	支持	订阅指定的通道并支持ConsumerRebalanceListener回调
void subscribe(Pattern var1, ConsumerRebalanceListener var2)	接口	支持	订阅所有匹配通配符的通道并支持ConsumerRebalanceListener回调
void unsubscribe()	接口	支持	取消所有订阅
ConsumerRecords<K, V> poll(long var1)	接口	支持	获取消息，但消息当中未实现checksum(消息的CRC32校验值)、serializedKeySize(key序列化后的字节长度)、serializedValueSize(key序列化后的字节长度)。

原生KafkaConsumer	类型	DISKafkaConsumer	说明
void commitSync()	接口	支持	同步提交当前消费的offset
void commitSync(final Map<TopicPartition, OffsetAndMetadata> offsets)	接口	支持	同步提交指定的offset
void commitAsync()	接口	支持	异步提交当前消费的offset
public void commitAsync(OffsetCommitCallback callback)	接口	支持	异步提交当前消费的offset并支持OffsetCommitCallback回调
void commitAsync(Map<TopicPartition, OffsetAndMetadata> offsets, OffsetCommitCallback callback)	接口	支持	异步提交指定的offset并支持OffsetCommitCallback回调
void seek(TopicPartition partition, long offset)	接口	支持	给分区设置指定的offset
void seekToBeginning(Collection<TopicPartition> partitions)	接口	支持	分区的offset设置为最旧的值
void seekToEnd(Collection<TopicPartition> partitions)	接口	支持	分区的offset设置为最新的值
long position(TopicPartition partition)	接口	支持	获取分区当前已消费数据的offset
OffsetAndMetadata committed(TopicPartition partition)	接口	支持	获取分区已提交的offset

原生KafkaConsumer	类型	DISKafkaConsumer	说明
List<PartitionInfo> partitionsFor(String topic)	接口	支持	获取通道的分区信息，但PartitionInfo里面的leader, replicas, inSyncReplicas未实现。
Map<String, List<PartitionInfo>> listTopics()	接口	支持	获取所有的通道信息，但PartitionInfo里面的leader, replicas, inSyncReplicas未实现。
void pause(Collection<TopicPartition> partitions)	接口	支持	暂停消费分区
void resume(Collection<TopicPartition> partitions)	接口	支持	恢复消费分区
Set<TopicPartition> paused()	接口	支持	获取所有已暂停消费的分区
close()	接口	支持	关闭consumer
Map<MetricName, ? extends Metric> metrics()	接口	不支持	获取统计信息
wakeup()	接口	不支持	内部实现原理不一样，不需要。
group.id	参数	支持	消费组ID
client.id	参数	支持	每个consumer的client.id必须唯一，如果不配置client.id, dis kafka consumer会生成一个uuid作为client.id。
key.deserializer	参数	支持	含义与kafka设置相同，但默认值为StringDeserializer (kafka必须配置)。
value.deserializer	参数	支持	含义与kafka设置相同，但默认值为StringDeserializer (kafka必须配置)。
enable.auto.commit	参数	支持	同kafka的默认设置，默认为true ● true表示启用自动提交，每隔${auto.commit.interval.ms}的时间提交一次offset; ● false表示不自动提交offset

原生KafkaConsumer	类型	DISKafkaConsumer	说明
auto.commit.interval.ms	参数	支持	自动提交offset的周期(毫秒)，默认值5000。
auto.offset.reset	参数	支持	同Kafka的默认配置，默认为latest。 此值用于没有初始偏移量或者偏移量不正确的情况下，自动设置offset位置： ● earliest 将偏移量自动重置为最旧的值; ● latest将偏移量自动重置为最新的值; ● none 如果在消费者组中没有发现前一个偏移量，就向消费者抛出一个异常;
其他参数	参数	不支持	-

6.3 使用 SDK（Python）

6.3.1 准备环境

- 已安装python2.7或以上版本，配置好python环境变量。
- 已安装pycharm。

6.3.2 配置样例工程

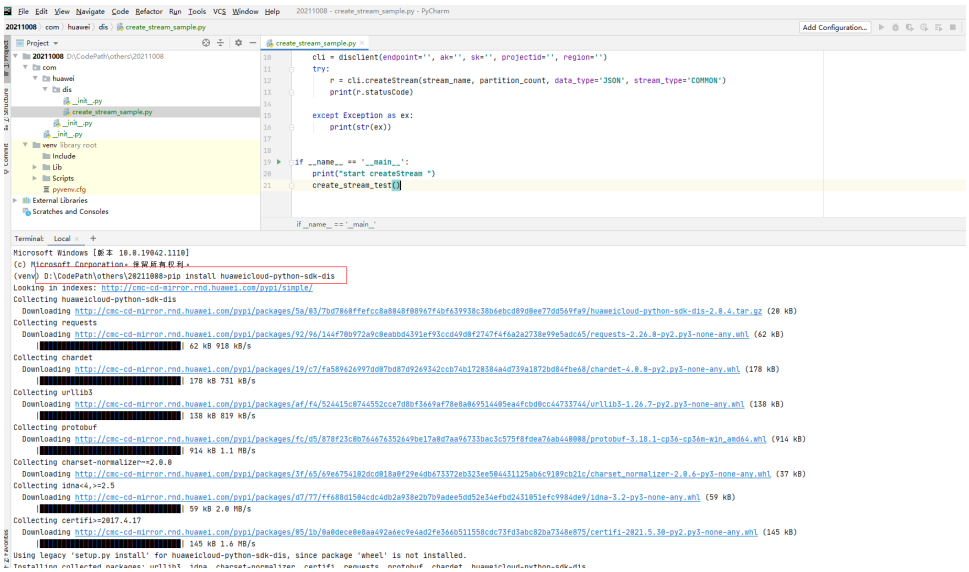
样例代码请查看：https://github.com/huaweicloud/huaweicloud-sdk-python-dis/tree/master/dis_sdk_python_demo。

操作步骤

步骤1 huaweicloud-python-sdk-dis已发布到 PyPi 之后，打开CMD，通过以下方式安装：pip install huaweicloud-python-sdk-dis。

步骤2 导入pycharm项目。

1. 打开pycharm。选择“File > open”弹出“Open File or Project”窗口。
2. 选择本地（可通过环境变量去查找）python安装目录下的“\Lib\site-packages\dis_sdk_python”样例工程的存储位置。（安装之后若未找到dis_sdk_python，请尝试升级pip，或者安装huaweicloud-python-sdk-dis）：

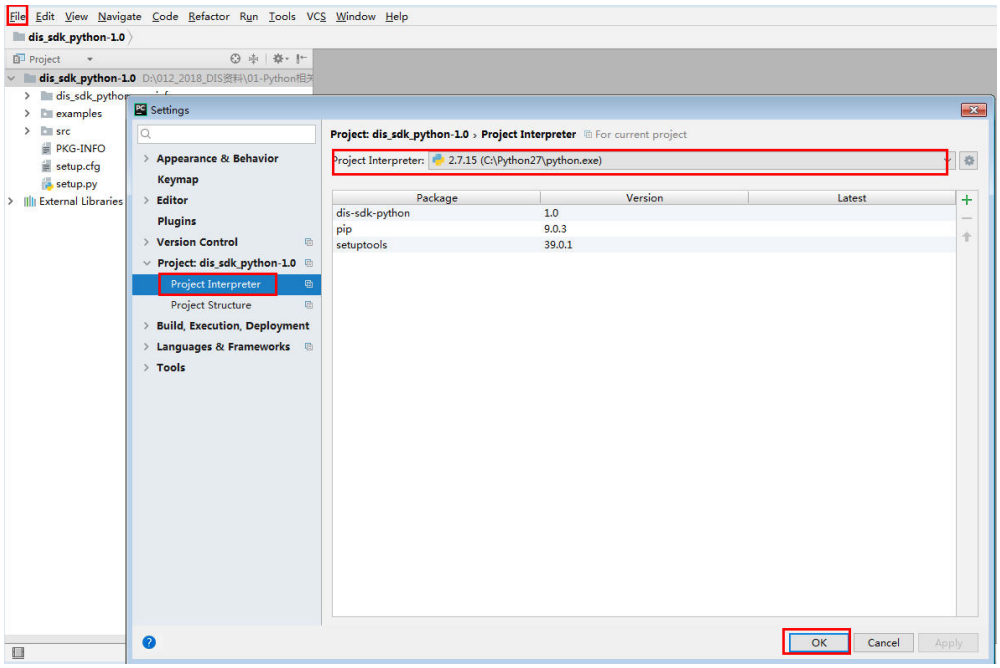


3. 单击“OK”完成项目导入。

步骤3 配置sdk_python工程。

1. 在左侧导航栏中选择“File > Settings > Editor > Font”，设置字体背景色等
2. 在左侧导航栏中选择“File > Settings > > Project Interpreter”“， 添加Python。
3. 选取合适的project Interpreter，单击“ok”。

图 6-4 添加 python



4. 在左侧导航栏中选择“File > Settings > Editor > File Encodings”， 设置pycharm编码。

Global Encoding、Project Encoding和Default encoding for properties files分别设置为UTF-8。

----结束

6.3.3 初始化 DIS 客户端

您可以使用以下方法初始化DIS SDK客户端实例。其中，“endpoint”，“ak”，“sk”，“region”，“projectId”信息请参考**获取认证信息**。

- ```
  cli = disclient(endpoint='**your-endpoint**',
                  // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
                  // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
                  ak=os.environ.get("HUAWEICLOUD_SDK_AK"),
                  sk=os.environ.get("HUAWEICLOUD_SDK_SK"),
                  projectid='**your-projectid**',
                  region='**your-region**')
  ```

6.3.4 创建通道

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

根据stream_type选取方法，配置方法中的参数值。

- stream_type=“” #无转储通道

配置createstream_sample.py中Dump_switch方法参数值。

- stream_type=“FILE” #文件类型通道

配置createstream_sample.py中Dump_switch_FILE方法参数值。

配置好参数后，执行createstream_sample.py文件默认调用createStream_test方法，获取响应201创建成功。

6.3.5 添加转储任务

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

参照配置方法中的参数值。

```
配置如下参数：
streamname='dis—test1'  #已存在的通道名
task_name='113'
```

以添加OBS转储服务为例：value参数值设定与key对应

```
basic_Schema=DumpTask.setSchema(key=['consumer_strategy','deliver_time_int
erval','agency_name','retry_duration'],
      value=['LATEST', 30, 'dis_admin_agency',1800])
obs_dump_task =['destination_file_type','obs_bucket_path','file_prefix', 'partition_format','record_delimiter']
obs_Schema = DumpTask.setSchema(basic_Schema=basic_Schema,
                key=obs_dump_task,value=['text','obs-1253', '','yyyy', '|'])
```

添加OBS转储服务,配置obs_Schema值

```
cli.add_dump_task(streamname, task_name,'OBS',obs_Schema)
```

- 配置好以上参数后，执行add_dump_task_sample.py文件默认调用add_dump_task_test方法，获取响应201创建成功。
```

6.3.6 删除通道

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname = "" #已存在的通道名称
```

配置好以上参数后，执行deleteStream_sample.py文件默认调用deleteStream_test方法，获取响应204删除成功。

6.3.7 删除转储任务

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname = "" #已存在的通道名称
task_name="xx"
```

说明

task_name配置为特定的转储任务名称，则删除通道下的该转储任务。

配置好以上参数后，执行delete_dump_task_sample.py文件默认调用delete_dump_task_test方法，获取响应204删除成功。

6.3.8 查询通道列表

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
start_stream_name = ""  #可设置为空，或是已存在的通道名
```

执行listStream_sample.py文件默认调用listStream_test方法，获取响应200查询成功。

通道列表的返回信息示例如下：

```
200
{'stream_names': ['dis-jLGp', 'dis-w_p', 'dis_test1', 'dis_test2'], 'has_more_streams': False, 'total_number': 4}
```

6.3.9 查询转储列表

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname="XXX" #已存在的通道名
```

执行list_dump_task_sample.py文件默认调用list_dump_task_test方法，获取响应200查询成功。

响应示例如下：

```
200
{'total_number': 1, 'tasks': [{'last_transfer_timestamp': 1538018769241, 'state': 'RUNNING', 'create_time': 1537949648144, 'destination_type': 'OBS', 'task_name': 'task_test1'}]}
```

6.3.10 查询通道详情

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname="dis-test1" #已存在的通道名
```

配置好以上参数后，执行describeStream_sample.py文件默认调用describeStream_test方法。

返回信息如下：

```
200
{"status": "RUNNING", "stream_name": "dis-test1", "data_type": "BLOB", "has_more_partitions": false, "stream_type": "COMMON", "stream_id": "L84hxfES223eVrFyxiE", "retention_period": 168, "create_time": 1532423353637, "last_modified_time": 1532423354625, "partitions": [{"status": "ACTIVE", "hash_range": "[0 : 9223372036854775807]", "sequence_number_range": "[0 : 10]", "partition_id": "shardId-0000000000"}]}
```

6.3.11 查询转储详情

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname="dis-test1"#已存在的通道名
task_name="test_1"  #查询该通道下的xx转储任务
```

配置好以上参数后，执行describe_dump_task_sample.py文件默认调用describe_dump_task_test方法。

返回信息如下：

```
200
{'state': 'RUNNING', 'stream_name': 'dis-test1', 'create_time': 1537949648144, 'last_transfer_timestamp': 1538018072564, 'destination_type': 'OBS', 'obs_destination_description': {'obs_bucket_path': '002', 'deliver_time_interval': 30, 'retry_duration': 0, 'agency_name': 'all', 'partition_format': 'yyyy/MM/dd/HH/mm', 'destination_file_type': 'text', 'record_delimiter': '|', 'consumer_strategy': 'LATEST', 'file_prefix': ''}, 'task_name': 'test_1', 'partitions': [{'state': 'RUNNING', 'discard': 0, 'last_transfer_offset': 500, 'partitionId': 'shardId-0000000000', 'last_transfer_timestamp': 1538018072564}, {'state': 'RUNNING', 'discard': 0, 'last_transfer_offset': 500, 'partitionId': 'shardId-0000000001', 'last_transfer_timestamp': 1538018072564}]}
```

6.3.12 Json 格式上传流式数据

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname="dis-test1"| #已存在的通道名
```

putRecords_sample.py文件中的putRecords_test方法中的records为需要上传的数据内容，数据上传格式如下：

```
records=[{"data": "abcdefd", "partition_id": “shardId-0000000001”}]
#"data": "xxx"为上传的数据值，请自定义；“partition_id”:“shardId-0000000001”为数据写入的分区id值，请自定义。
 record1 = {"data": "xxx","partition_id": partition_id}
#可写入多条数据，数据格式如record1所示，每写一条数据使用下面的append方法传入records中。
```

配置好以上参数后，执行putRecords_sample.py文件调用putRecords_test方法，响应结果如下：

```
200
{'failed_record_count': 0, 'records': [{'partition_id': 'shardId-0000000001', 'sequence_number': '15'}]}
```

6.3.13 Protobuf 格式上传流式数据

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

初始化DIS客户端，加入一项参数bodySerializeType，如下所示：

```
cli = disclient(endpoint='', ak=os.environ.get("HUAWEICLOUD_SDK_AK"),
sk=os.environ.get("HUAWEICLOUD_SDK_SK"), projectid='', region='',bodySerializeType='protobuf')
```

配置参数如下：

```
streamname="dis-test1" #已存在的通道名
```

参照该文件中的test方法，bodySerializeType="protobuf"选取protobuf格式上传。

protobuf_putRecords_sample.py文件中的protobuf_putRecords_test方法中的records为需要上传的数据内容，数据上传格式如下：

```
records=[{"data": "abcdefd", "partition_id": “shardId-0000000001”}]
#"data": "xxx"为上传的数据值，请自定义；“partition_id”:“shardId-0000000001”为数据写入的分区id值，请自定义。
 record1 = {"data": "xxx","partition_id": partition_id}
#可写入多条数据，数据格式如record1所示，每写一条数据使用下面的append方法传入records中。
```

配置好以上参数后，执行protobuf_putRecords_sample.py文件调用protobuf_putRecords_test方法，响应结果下：

```
200
{'failed_record_count': 0, 'records': [{'partition_id': 'shardId-0000000001', 'sequence_number': '15'}]}
```

6.3.14 下载流式数据

Json 格式下载

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname = "" #通道名称
startSeq='0’  #序列号
partitionId="shardId-0000000000"
```

通过getCursor_test更改游标类型cursorType='AT_SEQUENCE_NUMBER'

配置好以上参数，执行getRecords_sample.py文件调用getRecords_test方法，响应结果如下。

```
200
{'next_partition_cursor':
'eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzX3Rlc3QxIiwicGFydGl0aW9uLWlkIjoic2hhcmRJZC0wMDAwMDAwMDAwIiwiY3Vyc29yLXR5cGUiOiJBVF9TRVFVRU5DRV9OVU1CRVIiLCJzdGFydGluZy1zZXF1ZW5jZS1udW1iZXIiOiI2In0sImdlbmVyYXRlVGltZXN0YW1wIjoxNTU0NzA2NTc5MzA5fQ', 'records':
[{'sequence_number': '4', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558, 'timestamp_type':
'CreateTime'}, {'sequence_number': '5', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558,
'timestamp_type': 'CreateTime'}]}
```

Protobuf 格式下载

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

```
初始化DIS客户端，加入一项参数bodySerializeType，如下所示：
cli = disclient(endpoint='', ak=os.environ.get("HUAWEICLOUD_SDK_AK"),
sk=os.environ.get("HUAWEICLOUD_SDK_SK"), projectid='', region='',bodySerializeType='protobuf')
```

配置参数如下：

```
streamname = "" #通道名称
partitionId="shardId-0000000000"
```

通过getCursor_test更改游标类型cursorType='AT_SEQUENCE_NUMBER'

说明

获取游标getCursor_test采用test_0方法，下载数据getRecords_test采用test方法；test方法较test_0方法，增加参数bodySerializeType="protobuf"。

配置好以上参数，执行protobuf_getrecords_sample.py文件调用getRecords_test方法，响应结果如下。

```
200
{'next_partition_cursor':
'eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzX3Rlc3QxIiwicGFydGl0aW9uLWlkIjoic2hhcmRJZC0wMDAwMDAwMDAwIiwiY3Vyc29yLXR5cGUiOiJBVF9TRVFVRU5DRV9OVU1CRVIiLCJzdGFydGluZy1zZXF1ZW5jZS1udW1iZXIiOiI2In0sImdlbmVyYXRlVGltZXN0YW1wIjoxNTU0NzA2NTc5MzA5fQ', 'records':
[{'sequence_number': '4', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558, 'timestamp_type':
'CreateTime'}, {'sequence_number': '5', 'data': b'xxxxx', 'partitionKey': '0', 'timestamp': 1554705842558,
'timestamp_type': 'CreateTime'}]}
```

6.3.15 创建 APP

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
appName = "" #创建的APP名称
```

配置好以上参数，执行createApp_sample.py文件调用createApp_test方法，响应201表示创建成功。

6.3.16 删除 APP

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
appName = "" #待删除的APP名称
```

配置好以上参数，执行deleteApp_sample.py文件调用deleteApp_test方法，响应204表示删除成功。

6.3.17 查询 APP 详情

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
appname=”app1”      #查询的APP名称
```

配置好以上参数，执行describeApp_sample.py文件调用describeApp_test方法。

响应结果如下：

```
200
{'app_name': 'app1', 'app_id': 'OPKQuggQVtfqhyvK0cs', 'create_time': 1532425956631}
```

6.3.18 查询 APP 列表

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

listApp_test 方法中的（ limit可设置单次请求返回APP列表的最大数量取值范围：1~100 ）。

配置参数如下：

```
startAppName="app1"    #APP名称（从该通道开始返回app列表，返回的app列表不包括此app名称。）
```

配置好以上参数，执行listApp_sample.py文件调用Applist_test方法。

响应结果如下：

```
200
{'has_more_app': False, 'apps': [{'app_id': 'kpvGNrFYfKjpqTSdPIX', 'create_time': 1543301301992, 'app_name':
'sadfghjkl'}, {'app_id': 'MtPG1lD1E7IesDuOcNt', 'create_time': 1542765418080, 'app_name':
'testAppName2'}]}
```

6.3.19 新增 Checkpoint

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname = "" #通道名称
appName="xx"      # APP名称（APP是已存在状态）
partitionId="shardId-0000000000"   #分区的唯一标识符。
seqNumber="0"  #序列号
metadata=""   #用户消费程序端的元数据信息，元数据信息的最大长度为1000个字符
```

说明

partitionId可通过**查询通道详情**获取，需要先传入当前设置的通道名称。

配置好以上参数，执行commitCheckpoint_sample.py文件调用commitCheckpoint_test方法，响应201表示成功。

6.3.20 查询 Checkpoint

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname = "" #通道名称
appName="xx"      # APP名称（APP是已存在状态）
partitionId="shardId-0000000000"  #分区的唯一标识符
```

说明

partitionId可通过**查询通道详情**获取，需要先传入当前设置的通道名称。

配置好以上参数，执行getCheckpoint_sample.py文件调用getCheckpoint_test方法，响应结果如下：

```
{
  "sequence_number": "10",
  "metadata": "metadata"
}
```

6.3.21 变更分区数量

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
streamname = "" #已存在的running状态通道名
target_partition_count =”3”      #变更后的数量值
```

配置好以上参数，执行changepartitionQuantity_sample.py文件调用changepartitionQuantity_test方法，响应结果如下：

```
{
 "stream_name":"stream_name_test",
 "current_partition_count":2
 "target_partition_count":5
 }
```

6.3.22 获取数据游标

参见**初始化DIS客户端**的操作初始化一个DIS客户端实例。

配置参数如下：

```
partitionId="shardId-0000000000"
streamname=”dis-test1“ #已存在的通道名
5种游标设置使用参考如下：
# startSeq与AT_SEQUENCE_NUMBER/AFTER_SEQUENCE_NUMBER搭配使用
r = cli.getCursor(streamname, partitionId, cursorType='AT_SEQUENCE_NUMBER', startSeq="0")
# r = cli.getCursor(streamname, partitionId, cursorType='AFTER_SEQUENCE_NUMBER', startSeq="0")
# timestamp与AT_TIMESTAMP搭配使用
# r = cli.getCursor(streamname, partitionId, cursorType='AT_TIMESTAMP',timestamp=1554694135190)
# r = cli.getCursor(streamname, partitionId, cursorType='TRIM_HORIZON')
# r = cli.getCursor(streamname, partitionId, cursorType='LATEST')
```

配置好以上参数，执行getCursor_sample.py文件调用getCursor_test方法，响应结果示例如下：

```
200
{"partition_cursor":
"eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiSCIsInBhcnRpdGlvbi1pZCI6InNoYXJkSWQtMDAwMD
AwMDAwMCIsImN1cnNvci10eXBlIjoiQVRfU0VRVUVOQ0VfTlVNQkVSIiwic3RhcnRpbmctc2VxdWVuY2UtbnVt
YmVyIjoiMCJ9LCJnZW5lcmF0ZVRpbWVzdGFtcCI6MTUzMjQyNDg4NzE1NH0"}
```

7 异常信息

7.1 DIS 服务端错误码

在使用SDK进行操作时如果遇到错误，会在控制台显示错误码描述错误信息。

http状态码	错误码	Error Message	说明	处理措施
441	DIS.4100	Authorization error.	使用AKSK生成的签名信息错误	请检查请求头里的签名信息是否无误。
441	DIS.4101	Authorization header cannot be empty.	使用AKSK生成的签名信息为空	请求头里的签名信息为空，检查是否未生成签名信息。
441	DIS.4102	Incorrectly parsed authorization header.	无法解析签名	请检查请求头里的签名信息。
441	DIS.4103	Empty X-Sdk-Date header.	请求头里的X-Sdk-Date字段为空	请检查请求头里的X-Sdk-Date字段并补齐。
441	DIS.4104	Error parsing X-Sdk-Date header.	无法解析请求头里的X-Sdk-Date字段	请检查请求头里的X-Sdk-Date字段并修正。
441	DIS.4105	Invalid X-Sdk-Date header.	请求头里的X-Sdk-Date字段无效	请检查请求头里的X-Sdk-Date字段并修正。
441	DIS.4106	Empty AcessKey header.	请求头里的签名信息Authorization字段中缺失AK	请检查是否传入AK。

http状态码	错误码	Error Message	说明	处理措施
441	DIS.4107	Invalid AcessKey header.	请求头里的签名信息Authorization字段中的AK无效	请检查是否传入有效的AK，避免AK填写错误、AK被删除、临时AK过期等。
441	DIS.4108	Empty ServiceName header.	请求头里的签名信息Authorization字段中缺失服务名	请检查请求头里的Authorization字段中是否包含服务名dis。
441	DIS.4109	The Authorization header must contain the following field: {Credential,SignedHeaders,Signature;}	请求头里的签名信息Authorization字段有误	请检查请求头里的Authorization字段是否包含Credential，SignedHeaders，Signature。
441	DIS.4110	Empty Signature header.	请求头里的签名信息Authorization字段中没有SignedHeaders	请检查签名的生成方式是否有误。
441	DIS.4111	Invalid Region header.	请求头里的签名信息Authorization字段中的region无效	请检查是否传入有效的region。
441	DIS.4112	Invalid authorization request.	使用AKSK生成的签名信息错误	请检查签名的生成方式是否有误，检查AK、SK、region等信息。
441	DIS.4113	Empty Token header.	使用token认证时，请求头里的X-Auth-Token为空	请检查请求头里的X-Auth-Token。
441	DIS.4114	Invalid Token header.	使用token认证时，请求头里的X-Auth-Token无效	请检查请求头里的X-Auth-Token是否过期。
403	DIS.4116	Invalid RBAC.	用户操作受限	请根据返回的具体信息判断账号是否欠费、无DIS服务的操作权限等。

http状态码	错误码	Error Message	说明	处理措施
400	DIS.4117	Invalid Project Id.	用户传入的projectId无效	请检查传入的projectId是否有效，是否传入了其他project的id。
400	DIS.4200	Invalid request.	用户的请求无效	请参考API文档检查请求。
400	DIS.4201	Invalid partition_id.	用户传入的partition_id无效	请检查partition_id是否无效。
400	DIS.4202	Empty request.	用户的请求为空	请传入有效的请求。
400	DIS.4203	Invalid monitoring period.	查询监控信息的startTime无效	请传入有效的时间戳。
400	DIS.4204	The monitoring period cannot be longer than 7 days.	仅允许查询最近7天内的监控信息	请查询最近7天内的监控信息。
400	DIS.4208	Invalid MRS cluster.	创建MRS转储任务时，传入的MRS集群无效	请检查传入的MRS集群名称和ID，集群状态是否为运行中，以及是否为安全模式的集群。
400	DIS.4209	Invalid metrics label.	查询监控信息时，传入的监控指标不合法	请参考API文档检查监控指标并修正。
400	DIS.4215	Invalid cursor type.	获取数据游标时，传入的游标类型cursor-type不合法	请参考API文档检查cursor-type字段的范围并修正。
400	DIS.4216	Invalid sequence_number.	获取数据游标时，传入的序列号starting-sequence-number不合法	请传入有效的starting-sequence-number。
400	DIS.4217	Invalid partition cursor.	从DIS通道下载数据时，传入的数据游标partition-cursor无效	请重新获取partition-cursor并下载数据。
400	DIS.4219	The file is constantly resent.	该文件已经收到了	文件已经收到不需要再上传。

http状态码	错误码	Error Message	说明	处理措施
400	DIS.4220	The block whose sequence number is %s needs to be resent.	文件块需要重新上传	请按照指示上传对应的块。
400	DIS.4221	Block seq %s is expected	重复传入相同的文件块	请从系统期待的块开始上传。
400	DIS.4222	Block seq %s is expected.	传入的文件块不连续	从系统期待的块开始上传。
400	DIS.4223	The file size exceeds the limit.	文件的容量超过了DIS的限制	请拆分文件并再上传。
400	DIS.4224	The sequence number is out of range.	获取数据游标时，传入的序列号starting-sequence-number不在有效范围	请传入有效的starting-sequence-number。
400	DIS.4225	Expired partition cursor.	从DIS通道下载数据时，传入的数据游标partition-cursor过期	请重新获取partition-cursor并下载数据。
400	DIS.4226	A partition iterator error occurred or a record to which the SN corresponds has expired. Try to obtain the partition iterator again.	获取数据时，传入的数据游标partition-cursor对应的序列号starting-sequence-number过期	请获取获取数据游标，并用新游标获取数据。
400	DIS.4300	Request error.	请求体错误	请对照API文档修正请求体。
400	DIS.4301	The stream does not exist.	通道不存在	请检查传入的通道是否存在。
400	DIS.4302	The partition does not exist.	通道的分区不存在	请检查用户传入的分区ID是否存在。
400	DIS.4303	Exceeded traffic control limit.	超出流控	请扩容通道或降低上传速率。
400	DIS.4305	Too many stream requests.	同一时间内用户请求太多	请降低请求频率并重试。

http状态码	错误码	Error Message	说明	处理措施
400	DIS.4306	The bucket does not exist.	传入的OBS桶不存在	请检查OBS桶是否存在。
400	DIS.4307	The stream already exists.	指定的通道已经存在	请修改通道名称并重新创建通道
400	DIS.4308	Insufficient quota.	通道或分区的配额不足	请释放配额或提工单修改账号的配额。
400	DIS.4309	Too many request failures. Please try again later.	ip被加入黑名单	由于频繁的错误访问导致用户ip被加入黑名单，请检查认证信息和请求是否有效，并稍后重试。
400	DIS.4310	OBS access error.	访问OBS失败	请检查用户是否有访问OBS的权限。
400	DIS.4329	app quota exceeded.	APP配额超出限制	请释放APP的配额。
400	DIS.4330	app already exist.	已经存在同名的APP	请修改APP名称并重新创建APP。
400	DIS.4331	app is using.	删除app时，当前app在使用中	请确认app是否在使用中，如需删除请停止使用并重新删除。
400	DIS.4332	app not found.	指定的APP不存在	请检查指定的APP名称是否正确
400	DIS.4335	Invalid IAM agency.	创建转储任务时，使用的IAM委托无效	检查DIS创建的dis_admin_agency或用户自定义的IAM委托是否存在，权限是否完整。
400	DIS.4336	Invalid HDFS path.	创建MRS转储任务时，传入的MRS HDFS路径无效	请检查传入的MRS HDFS路径是否存在。
400	DIS.4337	The DLI database does not exist.	创建DLI转储任务时，传入的DLI数据库不存在	请检查传入的DLI数据库是否存在。
400	DIS.4338	The DLI table does not exist.	创建DLI转储任务时，传入的DLI数据表不存在	请检查传入的DLI表是否存在，并且是否为DLI内表。

http状态码	错误码	Error Message	说明	处理措施
400	DIS.4350	Invalid DWS cluster.	创建DWS转储任务时，传入的DWS集群不存在	请检查DWS集群是否存在，运行是否正常。
400	DIS.4351	Invalid KMS userKey.	创建DWS转储任务时，传入的KMS密钥信息无效	请检查KMS密钥是否存在。
400	DIS.4354	The transfer task does not exist.	删除或更新转储任务时，转储任务不存在	请检查转储任务是否存在。
400	DIS.4355	The transfer task already exists.	创建转储任务时，同一个通道下已存在同名的转储任务	请修改新创建转储任务的名称并重新创建。
400	DIS.4357	Exceeded transfer task quota.	单个通道仅允许同时存在5个转储任务，再创建新的转储任务会超出配额限制	请删除废弃的转储任务释放配额。
400	DIS.4358	The stream supports specific transfer tasks. Check the data type of the stream.	小文件转储的通道不支持创建普通转储任务	请创建新的通道并创建转储任务。
400	DIS.4360	Invalid data schema.	创建通道或更新通道时，传入的data_schema无效	请检查data_schema的格式并重试。
400	DIS.4601	The number of resource tags has reached the maximum.	一个资源上最多有10个标签，添加标签时资源上已添加的标签数超出限制	请删除废弃的标签并重新添加标签。
400	DIS.4602	Invalid resource type.	资源类型不合法	请检查资源类型是否合法。
400	DIS.4603	The resource does not exist.	资源不存在	请确认该资源是否已被删除。

http状态码	错误码	Error Message	说明	处理措施
400	DIS.4604	The key does not exist.	标签Key不存在	请确认标签Key是否存在。
400	DIS.4605	The action is not supported.	当前标签操作不支持	请确认当前标签操作是否合法，当前仅支持create和delete操作。
500	DIS.5000	System error.	内部服务错误	请联系技术支持。
500	DIS.5100	HBase error.	连接HBase超时、转储查询异常、其他HBase异常	请联系技术支持。
500	DIS.5150	Redis error.	Redis连接异常、数据异常、消息广播异常deng	请联系技术支持。
500	DIS.5200	Zookeeper error.	Zookeeper异常，创建topic、删除topic、 给topic添加分区时失败	请联系技术支持。
500	DIS.5250	Kafka error.	Kafka异常	请联系技术支持。
500	DIS.5251	Kafka create topic timeout.	Kafka创建topic超时	请联系技术支持。
500	DIS.5252	Kafka topic does not exist.	Kafka连接错误	请联系技术支持。
500	DIS.5380	Kafka Connect error.	Kafka连接错误	请联系技术支持。
500	DIS.5400	Resourcemgt error.	Resourcemgt服务创建、更新、删除通道错误或新增分区错误	请联系技术支持。
500	DIS.5401	Kafka partition resource exhausts.	Kafka分区资源耗尽	请联系技术支持。
500	DIS.5402	Kafka partition resources are about to be exhausted.	Kafka分区资源容量即将售罄	请联系技术支持。

http状态码	错误码	Error Message	说明	处理措施
500	DIS.5550	Firehose error.	创建文件失败或者传输文件到OBS失败	请联系技术支持。
500	DIS.5600	Service admin account error.	内置租户账号异常	请联系技术支持。
500	DIS.5601	Service op svc account error.	管理租户账号异常	请联系技术支持。
500	DIS.5750	IAM error.	服务内部调用IAM服务异常	请联系技术支持。
500	DIS.5760	CES error.	将指标上载到CES错误	请联系技术支持。
500	DIS.5780	DCS error.	Redis链接失败	请联系技术支持。
500	DIS.5850	OBS error.	OBS异常	请联系技术支持。
500	DIS.5900	Partition is readonly for DISK is not enough.	磁盘已满，无法写入数据	请联系技术支持。

A 修订记录

发布日期	修订说明
2019-05-08	第一次正式发布。